Jacek Matulewski
http://www.phys.uni.torun.pl/~jacek/

Podstawy programowania RAD z użyciem narzędzi Borland Delphi/C++ Builder

Ćwiczenia

Toruń, 21 marca 2003

Najnowsza wersja tego dokumentu znajduje się pod adresem
http://www.phys.uni.torun.pl/~jacek/dydaktyka/rad/rad1.pdf

Źródła opisanych w tym dokumencie programów znajdują się pod adresem
http://www.phys.uni.torun.pl/~jacek/dydaktyka/rad/rad1.zip

I. Spis treści

II. Podstawy programowania RAD

1. Pojęcia podstawowe

RAD = Rapid Application Development (błyskawiczne tworzenie aplikacji)
C++ Builder – zintegrowane środowisko programistyczne (edytor + kompilator + debugger) przeznaczone dla systemu Microsoft Windows 9*/ME i NT/2000/XP oparte na języku C++
Delphi – j.w., ale w oparciu o Object Pascal
komponenty – obiekty (pochodne klasy TComponent) zarejestrowane w środowisku C++ Builder/Delphi i dostępne na palecie komponentów w trakcie projektowania aplikacji
VCL = Visual Component Library – biblioteka komponentów dostarczanych przez Borland
Microsoft Visual Basic – konkurencja na rynku RAD.

2. Z czego składa się „pusty" projekt?

Funkcjonalność aplikacji stworzonej przez środowisko: Skompilować „pusty" projekt. Sprawdzić działanie aplikacji (zmiana rozmiaru okna, przenoszenie, zamykanie, itp.)
Rodzaje form: BorderStyle, FormStyle, Caption, Arrow, itp.

3. Podstawy projektowania RAD

TLabel: Dodać do projektu formy obiektu klasy TLabel i edytować jego własności (Caption, Caption z accel char np. &Zamknij, Color, Font, Visible).
TButton: Analogicznie umieścić na formie Button1 i edytować własności oraz metodę zdarzeniową związanej ze zdarzeniem OnClick (zamknięcie aplikacji przez `Close()` i przez `Application->Terminate()`). Zmiana własności Label1 podczas działania programu.

4. TImage i TOpenDialog

TImage: Wrzucić na formę obiekt klasy TImage (zakładka VCL: Additional) i wczytać bitmapę. Dodać klawisz i w metodzie zdarzeniowej obsłużyć czytanie obrazu z pliku funkcją
`Image1->Picture->LoadFromFile()`.
TOpenDialog: Poprawić projekt używając okna dialogowego do wyboru pliku (należy zadbać, aby pozwolić na czytanie tylko z istniejącego pliku).

Zadanie: zastąpić TImage komponentem TMemo i korzystając z TOpenDialog i TSaveDialog przygotować aplikację analogiczną do Notatnika w Windows.

7. Menu główne, kontekstowe

Edytor menu głównego i menu kontekstowego. Wielopoziomowe, złożone menu. Do poprzedniej aplikacji dodać menu główne (Plik\Wczytaj..., Zapisz.... oraz Zamknij).

5. Dodawanie form do projektu

Dodać formę do projektu. Pojęcie **formy główna**. Własność `TApplication->ShowMainForm`. Różnica między efektami metod `TForm->Show()` i `ShowModal()`.

6. Komponenty sterujące

Na Form2 z poprzedniego punktu umieść dwa suwaki klasy TScrollBar sterujące rozmiarem Form1 oraz `TForm2->Edit1` sterujący własnością `TForm1->Caption`. Określanie własności `TScrollBar->Min`, Max i korzystanie z własności `Position`. Dołączenie `ProgressBar` odzwierciedlającego szerokość lub wysokość formy (`Form->OnResize`)

8. Pliki projektu:

C++ Builder	Delphi	Opis
*.mak lub *.bpr	*.dpr	Główny plik projektu (rozszerzenie zależy od wersji Buildera)
	*.dfo	Informacje o opcjach projektu (w C++ Builderze informacje te umieszczone są w pliku projektu)
*.cpp i *.h	*.pas	Pliki z kodem C++; osobny plik o tej samej nazwie, co projekt i po jednym dla każdej formy; ponadto użytkownik może pisać i dodawać do projektu własne moduły
*.dfm	*.dfm	Pliki, w których Builder przechowuje informacje o zaprojektowanych przez użytkownika elementach formy (w obu środowiskach format tych plików jest identyczny)
*.ilc, *.ild, *.ilf, *.ils, *.tds		Pliki tworzone przez kompilator dla przyspieszenia kompilacji – można je bez straty skasować
*.obj	*.dcu	Skompilowane formy i inne objekty tworzone przez użytkownika w oddzielnych plikach; można je skasować jeżeli posiadamy źródła
*.res	*.res, *.dcr	Pliki zasobów (format plików *.res jest identyczny w obu środowiskach); plik *.dcr przechowuje zasoby (bitmapy, kursowy), których wymaga tworzony przez użytkownika komponent – np. ikonę jaka pojawi się na palecie.
.~	*.~*	Backupy edytora – można je również kasować

Warto przygotować sobie plik wsadowy (C++ Builder):

```
@echo off
del *.~*
del *.bak
del *.ilc
del *.ild
del *.ilf
del *.ils
del *.obj
del *.tds
del *.dat
del *.exe
```

9. Owner vs. Parent

Konwencja: listingi zielone dotyczą C++ Buildera, a niebieskie Delphi.

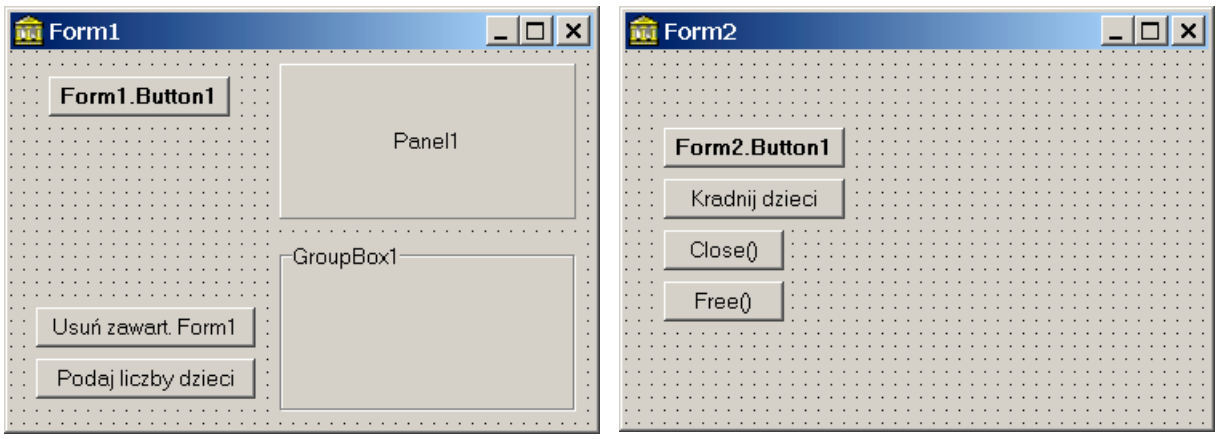
`Komponent1`.**Owner** – wskazuje na właściciela, tj. komponent odpowiedzialny za zwolnienie pamięci zajmowanej przez `Komponent1`. Jest to zazwyczaj obiekt, w którego deklaracji znajduje się wskaźnik do `Komponent1` (np. Form1). Usunięcie z pamięci właściciela spowoduje usunięcie także wszystkich obiektów, którego jest właścicielem.

`Komponent1`.**Parent** – wskazuje na rodzica, tj. obiekt wewnątrz którego znajduje się (czyli po prostu „narysowany jest") `Komponent1` (mogą to być komponenty typu panel, grupa, forma i inne pochodne względem TWinControl). Zmiana położenia rodzica spowoduje odpowiednią zmianę położenia dziecka.

Każdy komponent dziedziczący po TComponent zawiera listę wszystkich komponentów, których jest właścicielem Components (warto również zwrócić uwagę na własność ComponentCount, która przechowuje informację o ilości komponentów w tej liście), a komponent pochodny TWinControl posiada listę swoich dzieci Controls (i analogicznie ControlCount).

Uwaga! Pewne zamieszanie wprowadza używanie terminu „rodzic" przy opisie dziedziczenia klas. Ani Owner, ani Parent nie ma z tym nic wspólnego.

Przykład dynamicznej zmiany własności Parent i wykorzystywania listy Components
Stwórzmy dwie formy wg wzoru:



Do przycisków „Close” i „Free” na Form2 podłączmy metody wywołujące odpowiednio ukrycie formy i usunięcie formy z pamięci.

Początkowo będą nas interesować dwa przyciski z wytłuszczonymi etykietami. Chcemy, aby kliknięcie formy, panelu i groupboxa zmieniało właściciela tych przycisków na kliknięte obiekty. W tym celu musimy ze zdarzeniem OnClick każdego z tych obiektów związać metodę zawierającą polecenia typu (przykład dla Panel1Click):

```
Form1.Button1.Parent:=Panel1;
Form2.Button1.Parent:=Panel1;
```

Oczywiście w metodach odnoszących się do innych obiektów musi zmienić się wartość przypisana do własności Parent. Po skompilowaniu możemy zobaczyć, że przyciski można przenosić nie tylko w obrębie formy, ale i pomiędzy dwoma formami tej samej aplikacji. Przyciski zachowują swoją wielkość i relatywne położenie względem górnego lewego rogu rodzica, więc jeżeli przycisk przenosimy np. z formy na mniejszy od niej panel może się zdarzyć, że położenie przycisku spowoduje, że nie mieszcząc się w obrębie panelu nie będzie on w ogóle widoczny.

Wykorzystując klawisz „Free” można się przekonać, że przycisk „Form2.Button1” zostanie usunięty wraz z Form2, nawet jeżeli aktualnym rodzicem jest Form1. (Po usunięciu Form2 kliknięcie panelu, grupy lub Form2 spowoduje błąd, gdyż nastąpi odwołanie do Form2.Button2, który już nie istnieje.)

Do klawisza „Podaj liczby dzieci” (osobno dla każdej z form) podłączmy metodę, która wyświetli na etykiecie formy liczbę obiektów, których każda z form jest właścicielem i rodzicem:

```
procedure TForm1.Button2Click(Sender: TObject);
var S :String;
begin
//Rodzic (Parent)
Str(Form1.ControlCount,S); Form1.Caption:='Parent:'+S;
Str(Form2.ControlCount,S); Form2.Caption:='Parent:'+S;
//Właściciel (Owner)
Str(Form1.ComponentCount,S); Form1.Caption:=Form1.Caption+', Owner:'+S;
Str(Form2.ComponentCount,S); Form2.Caption:=Form2.Caption+', Owner:'+S;
end;
```

Z kolei klawisz „Kradnij dzieci” będzie służyć do zmiany rodzica komponentów, których obecnym rodzicem jest Form1:

```
procedure TForm2.Button4Click(Sender: TObject);
var i :Integer;
begin
for i:=0 to Form1.ControlCount-1 do Form1.Controls[0].Parent:=Form2;
            //Ustawienie w tablicy Controls zmienia sie po kazdej
```

```
            //kradziezy, dlatego ciagle pobieram pierwszy element z listy
            //(można również zastosować pętle z downto)
end;
```

Wykorzystujemy tu listę Form1.Controls, listę komponentów, których rodzicem jest Form1.
Można również wykorzystać listę Form1.Components, np. w celu skasowania wszystkich obiektów, których właścicielem jest Form1 bez względu na to gdzie się znajdują (klawisz „Usuń zawartość Form1”):

```
procedure TForm1.Button3Click(Sender: TObject);
var i :Integer;
begin
for i:=Form1.ComponentCount-1 downto 0 do
    begin
    if Form1.Components[i]<>Button3 then Form1.Components[i].Free();
    end;
end;
```

Przy usuwaniu pomijamy Button3, którym wywołujemy usunięcie. Aby uniknąć błędu odwoływania do Form1.Button1 w metodach zmieniających rodziców można również pominąć ten przycisk.

10. Dynamiczne tworzenie obiektów VCL

Umieścić w metodzie `Form1->OnMouseDown` następujący kod:

```
//C++ Builder
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
TButton* DynamicButton=new TButton(this); //dynamiczne tworzenie obiektu
DynamicButton->Width=110;
DynamicButton->Left=X;
DynamicButton->Top=Y;
DynamicButton->Enabled=false;
switch(Button)
      {
      case mbLeft: DynamicButton->Caption="Lewy"; break;
      case mbRight: DynamicButton->Caption="Prawy"; break;
    case mbMiddle: DynamicButton->Caption="środk."; break;
    default: DynamicButton->Caption="???"; break;
    }
DynamicButton->Caption=DynamicButton->Caption+" ("+X+","+Y+")";
DynamicButton->Parent=this;
}

//Delphi
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var DynamicButton :TButton; //TButton wymaga dodania modułu stdctrls
    XStr,YStr :string;
begin
DynamicButton:=TButton.Create(Self); //dynamiczne tworzenie obiektu
DynamicButton.Width:=110;
DynamicButton.Left:=X;
DynamicButton.Top:=Y;
DynamicButton.Enabled:=false;
case Button of
     mbLeft: DynamicButton.Caption:='Lewy';
     mbRight: DynamicButton.Caption:='Prawy';
     mbMiddle: DynamicButton.Caption:='środk.';
     end;
```

```
Str(X,XStr);
Str(Y,YStr);
DynamicButton.Caption:=DynamicButton.Caption+' ('+XStr+','+YStr+')';
DynamicButton.Parent:=Self;
end;
```

Powyższy kod wymaga dodania biblioteki stdctrls w Delphi. W zaznaczonej komentarzem linni następuje powołanie nowego obiektu przypisanego do zmiennej DynamicButton w standardowy sposób. Następnie ustalane są jego własności geometryczne oraz napis uzależniony od naciśniętego klawisza myszy i wreszcie zostaje pokazany przez ustalenie rodzica na obiekt macierzysty, czyli Form1.

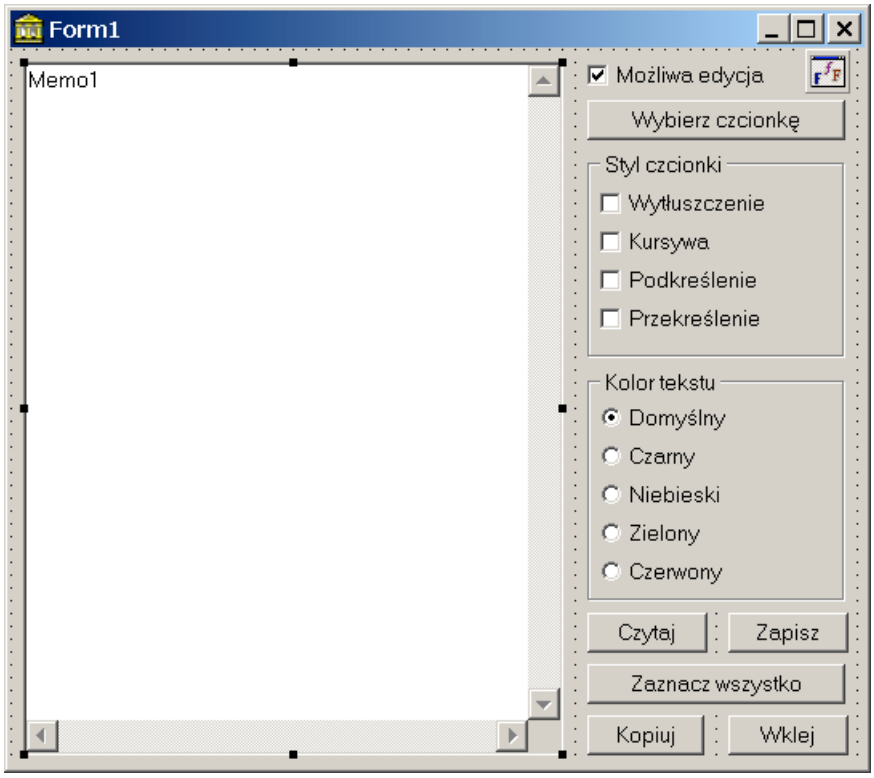
Zadanie
Zmodyfikować program tak, żeby wskaźniki do tworzonych dynamicznie obiektów umieszczać w tablicy (lub lepiej w stosie). W metodzie OnDestroy, OnClose lub OnCloseQuery umieścić procedurę, która zwolni pamięć używaną przez te obiekty. Można także wykorzystać listę komponentów znajdujących się na formie (`Form1->Controls)` lub komponentów, których forma jest właścicielem (`Form1->Components`)

Zadanie (C++ Builder)
Jak zmieni się działanie programu jeżeli do deklaracji zmiennej DynamicButton dodamy słowo **static**:
```
static TButton* DynamicButton=new TButton(this);
```

III. Przykład wykorzystania komponentów VCL do szybkiego projektowania aplikacji

1. Notatnik zaprojektowany w oparciu o komponent TMemo

Stworzymy aplikację pełniącą formę notesu, w którym (w bardzo ograniczonym zakresie) będziemy mogli modyfikować sposób wyświetlania tekstu na ekranie. Na formę położmy obiekt typu TMemo oraz obiekty TGroupBox, TRadioGroup i kilka przycisków jak pokazano poniżej. W Memo1 za pomocą Object Inspectora ustalono opcję ScrollBars na ssBoth.



Następnie będziemy kolejno oprogramowywali widoczne na formie komponenty kontroli.

a) „Możliwa edycja”
Ten CheckBox ma kontrolować możliwość zmieniania przez użytkownika zawartości Memo1.

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
Memo1.Enabled:=CheckBox1.Checked;
end;
```

b) „Wybierz czcionkę”
Tym przyciskiem uruchomimy standardowe okno wyboru czcionki i jej stylu oraz zastosujemy nowo wybraną czcionkę do okna Memo1.

```
procedure TForm1.Button6Click(Sender: TObject);
begin
if FontDialog1.Execute then Memo1.Font:=FontDialog1.Font;
end;
```

Metoda Execute zwraca wartość True, jeżeli użytkownik wybrał czcionkę i nacisnął OK.

c) „Styl czcionki”
Ręczny wybór stylu czcionki – trudność polega na sposobie określania stylu jako zmiennej kolektywnej (czyt. zbioru). Należy zadeklarować zmienną `styl` przechowującą czcionki typu TFontStyles. Typ ten jest zbudowany w oparciu o tyb wyliczeniowy TFontStyle wymieniający cztery typy formatowania: fsBold, fsItalic, fsUnderline, fsStrikeOut. Obsługa ComboBoxów zgrupowanych w GroupBox1 polegać będzie na dodawaniu lub usuwaniu ze zbioru `styl` odpowiednich elementów.

```
//Delphi
procedure TForm1.CheckBox2Click(Sender: TObject);
var styl :TFontStyles;
begin
styl:=Memo1.Font.Style;
if CheckBox2.Checked then Include(styl,fsBold) else Exclude(styl,fsBold);
if CheckBox3.Checked then Include(styl,fsItalic) else Exclude(styl,fsItalic);
if CheckBox4.Checked then Include(styl,fsUnderline) else
                                                    Exclude(styl,fsUnderline);
if CheckBox5.Checked then Include(styl,fsStrikeout) else
                                                    Exclude(styl,fsStrikeout);
Memo1.Font.Style:=styl;
end;
```

Tutaj zastosowałem jedną metodę zdarzeniową związaną z wszystkimi CheckBoxami z grupy, ale z powodzeniem można napisać oddzielną dla każdego z nich. Być może wówczas wygodniejsze byłoby zadeklarowanie zbioru styl globalnie dla całej formy – należy wtedy pamiętać o jej uaktualnieniu także przy zmianie czcionki za pomocą okna dialogowego.

d) „Kolor tekstu”
RadioButtony wewnątrz RadioGroup nie są komponentami zrzuconymi myszką z panelu – tworzy się je tutaj w edytorze własności RadioGroup1.Items. Zaznaczony element wybiera się za pomocą własności ItemIndex (numerowanej jak w C od 0). Od tej własności uzależnimy też kolor czcionki w Memo1:

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
with Memo1.Font do
case RadioGroup1.ItemIndex of
     0: Color:=clWindowText;
     1: Color:=clBlack;
     2: Color:=clNavy;
     3: Color:=clGreen;
     4: Color:=clRed;
     end;
end;
```

Bardzo ciekawą, specyficzną dla Object Pascala, jest konstrukcja `with` *`obiekt`* `do ...;`, w której zamiast kropek można pisać własności i metody *obiektu* bez wymieniania całej struktury dostępowej. Równoważne jest zatem napisanie

```
Memo1.Font.Color:=clMaroon;
Memo1.Font.Size:=10;
```

oraz

```
with Memo1.Font do
      begin
      Color:=clMaroon;
      Size:=10;
      end;
```

e) „Czytaj" i „Zapisz"
TMemo ma własność Lines typu TStrings, która z kolei posiada metody służące do czytania z i zapisu do pliku tekstowego. Są to Memo1.Lines.LoadFromFile(*nazwa pliku*) i Memo1.Lines.SaveToFile(*nazwa pliku*).

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Memo1.Lines.LoadFromFile('notes.txt');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Memo1.Lines.SaveToFile('notes.txt');
end;
```

Próba czytania z pliku 'notes.txt' przed jego stworzeniem skończy się błędem.

f) „Zaznacz wszystko"
To jest chyba najprostsze zadanie:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
Memo1.SelectAll;
Memo1.SetFocus; //to polecenie zwraca "focus" do Memo1
end;
```

Druga linia metody zwraca „focus" do okna edycyjnego. Inaczej nie widać byłoby zaznaczenia tekstu (chyba, że opcja Memo1.HideSelection ustawiona jest na False).

g) „Kopiuj" i „Wklej" – współpraca ze schowkiem
Informację zawartą w schowku można obejrzeć wykorzystując komponent TClipboard, ale tu nie jest to konieczne, gdyż TMemo zawiera metody, które kopiowanie i wklejanie ze schowka wykonają za nas: CopyToClipboard, CutToClipboard oraz PasteFromClipboard. W istocie jest to dublowanie funkcji dostępnych już w Memo pod typowymi skrótami klawiszy (Ctrl+C, Ctrl+X, Ctrl+V).

```
procedure TForm1.Button4Click(Sender: TObject);
begin
Memo1.CopyToClipboard;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
Memo1.PasteFromClipboard;
end;
```

Zadanie 1
Do formy dodaj komponenty dialogów TOpenDialog i TSaveDialog. Następnie w metodach związanych z zapisem i czytaniem tekstu do pliku dodaj możliwość wyboru nazwy pliku i jego ścieżki. W oknie TOpenDialog można tak wybrać opcje, aby niemożliwe było czytanie pliku, który nie istnieje.

Zadanie 2
W przypadku braku zaznaczonego fragmentu tekstu przy próbie skopiowania zgłoś odpowiedni komunikat.

Zadanie 3
Zastąp komponent TMemo komponentem TRichEdit. Wszystkie formatowania powinny odnosić się do **TRichEdit.SelAttributes**. Przypisanie TFont do tego obiektu można zrobić korzystając z funkcji Assign np. `RichEdit1.SelAttributes.Assign(FontDialog1.Font)`. Panel z prawej strony można zastąpić przez pasek narzędziowy lub menu. Rozwiązanie znajduje się w źródłach dołączonych do dokumentu.

IV. Grafika w C++ Builder/Delphi

Podstawowe pomysły niektórych przykładów z tego rozdziału zostały zaczerpnięte
z książki Andrzeja Stasiewicza *C++ Builder – całkiem inny świat*.

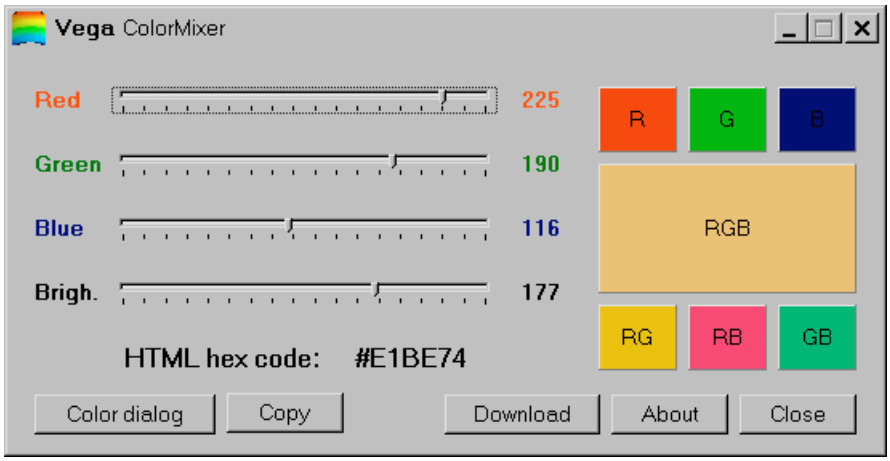
Konwencja: listingi zielone dotyczą C++ Buildera, a niebieskie Delphi.

1. Kolory

Własność TColor (Form1.Color, Panel1.Color itd)
Funkcja WinAPI `unsigned long RGB(int,int,int)`

Za pomocą trzech suwaków będziemy kontrolować kolor panelu zmieniając jego kolory składowe RGB.

Na formie nowego projektu połóżmy trzy przewijalne paski typu TrackBar (paleta Win95), opiszmy je za pomocą Label jako R, G i B. Ponadto umieśćmy trzy małe i jeden duży komponenty Panel z palety standardowej, w których prezentować będziemy kolory odpowiadające pozycjom poszczególnych pasków i kolor ostateczny.



Ilustracja pokazuje bardziej rozwiniętą wersję tego projektu.

Informacji o funkcjach WinAPI, w tym o funkcji RGB nie znajdziemy w plikach pomocy C++ Buildera czy Delphi. Znajdują się one w plikach MS Help (wersja Proffesional i Enterprise).
Funkcja RGB przyjmuje trzy argumenty typu int o wartości mniejszej 256, a zwraca trzybajtowy kod koloru.

Można więc napisać polecenia (najlepiej w metodzie związanej ze zdarzeniem TrackBar1->OnChange

```
Panel1->Color=(TColor)RGB(TrackBar1->Position,0,0);
Panel2->Color=(TColor)RGB(0,TrackBar2->Position,0);
Panel3->Color=(TColor)RGB(0,0,TrackBar3->Position);
Panel4->Color=(TColor)RGB(TrackBar1->Position,TrackBar2->Position,
                                              TrackBar3->Position);
```

Analogiczne polecenia w Delphi:

```
Panel1.Color:=RGB(TrackBar1.Position,0,0);
Panel2.Color:=RGB(0,TrackBar2.Position,0);
Panel3.Color:=RGB(0,0,TrackBar3.Position);
Panel4.Color:=RGB(TrackBar1.Position,TrackBar2.Position,TrackBar3.Position);
```

Jest to oczywiście za dużo, bo po zmianie TrackBar1 nie trzeba zmieniać koloru paneli związanych jedynie z pozostałymi suwakami (Panel2 i Panel3), natomiast możemy tę samą metodę związać z TrackBar2->OnChange

Pomijamy wszelkie aspekty estetyczne (np. rozmieszczenie znaczników przy suwakach), ale, żeby aplikacja działała prawidłowo należy zmienić zakres jaki może przyjmować TrackBar->Position na (0, 255), wobec tego niech TrackBar->Max=255. Warto też już w trakcie projektowania uzgodnić kolory paneli z początkowymi położeniami suwaków, tj. zmienić wszystkie kolory na czarne.

TColorDialog
Na palecie Dialogs znajduje się komponent TColorDialog, który obsługuje typowe dla Windows okienko wyboru koloru (znane chociażby z Painta). Dodajmy do naszej formy ten komponent. Jego uruchomienie i wybór koloru powinien spowodować zmianę koloru dużego panelu na wybrany, a małych na odpowiednie kolory składowe. Również pozycje suwaków powinny się odpowiednio zmienić.
Dodajmy do formy przycisk, którym użytkownik będzie otwierał okno dialogowe.
W metodzie związanej ze zdarzeniem OnClick wpiszmy kod:

```
ColorDialog1->Color=Panel4->Color;
if (ColorDialog1->Execute())
      {
      TrackBar1->Position=GetRValue(ColorDialog1->Color);
      TrackBar2->Position=GetGValue(ColorDialog1->Color);
      TrackBar3->Position=GetBValue(ColorDialog1->Color);
      TrackBar1Change(Sender);
      };
```

W pierwszej linii uzgadniamy kolor wybrany w oknie dialogowym, z kolorem dużego panelu.
Druga linia uruchamia modalne okno o dialogowe.
Trzy kolejne linie czytają składowe wybranego przez użytkownika koloru (dzięki pierwszej linii nie trzeba dbać o możliwość anulowania wyboru) i przypisuje odpowiednie pozycje suwakom.
Czwarta linia uruchamia metodę, która na podstawie pozycji suwaków ustalała kolory paneli.

Aby uatrakcyjnić wygląd okna dialogowego można w inspektorze obiektów ustalić wartość własności `ColorDialog1->Options->cdFullOpen` na `True`. W istocie Options to zbiór, którego elementem może być cdFullOpen, więc z poziomu kodu analogiczne polecenie miałoby postać:

```
ColorDialog1->Options << cdFullOpen;
```

lub

```
ColorDialog1->Options << cdFullOpen << kolejne_elementy;
```

W Delphi:

```
Include(ColorDialog1.Options,cdFullOpen);
```

ewentualnie

```
ColorDialog1.Options:=ColorDialog1.Options+[cdFullOpen,kolejne_elementy];
```

Dodatek: HTML color hex code
Za pomocą funkcji standardowej (stdlib.h) C++ `itoa(liczba,łańcuch,16)`[1] można dla każdego suwaka przekonwertować liczbę na łańcuch i po zsumowaniu przypisać do, powiedzmy, Label4 kod koloru gotowy do wstawienia do dokumentu HTML.
Za pomocą kilku poleceń można ten kod umieścić w schowku (należy pamiętać, żeby dołączyć odpowiednią bibliotekę, w tym przypadku clipbrd.h):

[1] W HTML kolory kodowane są trójką liczb szesnastkowych (np. #000000 to kolor czarny, #00FF00 to kolor zielony i #FFFFFF – biały).

```
TClipboard* Clipboard;
Clipboard = new TClipboard;
Clipboard->Clear();
Clipboard->SetTextBuf(Label4->Caption.c_str());
Clipboard->Close();
delete Clipboard;
```

Wersja Delphi:

```
procedure TForm1.Button1Click(Sender: TObject);
var Clipboard :TClipboard;
begin
Clipboard:=TClipboard.Create();
Clipboard.Clear();
Clipboard.SetTextBuf(PChar(Label4.Caption));
Clipboard.Close();
Clipboard.Free();
end;
```

Dodatek: AlphaBlend
W Delphi/C++ Builder w wersji 6 lub nowszych można manipulować przezroczystością formy. Daje to bardzo interesujący efekt wizualny.
Umieśćmy na formie dodatkowy pasek `TTrackBar` z wyznaczonym zakresem od 100 do 255 i pozycją ustaloną na 255. Przestawmy własność formy `AlphaBlend` na true i w zdarzeniu `TTrackBar.OnChange` powiążmy własność formy `AlphaBlendValue` z pozycją kontrolki.

2. Płótno formy (Canvas) – kreślenie linii, pióra

Wiele obiektów, m.in. formy, posiadają własność typu TCanvas (np. Form1.Canvas). Canvas gromadzi narzędzia do rysowania zarówno figur geometrycznych, jak i ustalania kolorów poszczególnych pikseli. Jej metody pozwalają na rysowanie figur geometrycznych (linia, prostokąt, elipsa, łuk, wycinek koła itp.).

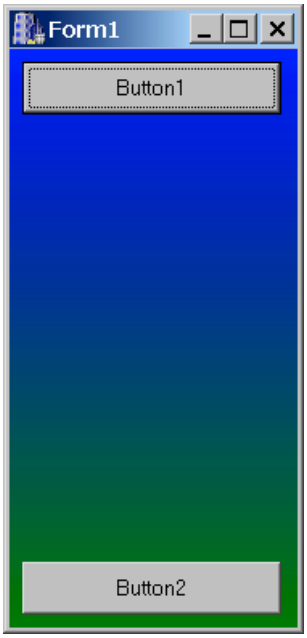
Pasek gradientowy o dowolnie ustalanych kolorach
Zadeklarujmy dwie zmienne typu TColor w sekcji private klasy naszej formy (aby dostać się do pliku nagłówkowego Unit1.h wystarczy nacisnąć Ctrl+F6). Niech zmiene te nazywają się Kolor1 i Kolor2.

```
TColor Kolor1;
TColor Kolor2;
```

Należy zainicjować wartość zmiennej (najlepiej w kreatorze Form1), ponieważ ani Pascal, ani C++ o to nie dba.

```
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
Kolor1=clGreen;
Kolor2=clBlue;
}
```

Na formie umieścmy dwa przyciski i komponent ColorDialog.



Oba przyciski będą wywoływać ten sam komponent ColorDialog1. Odpowiednie metody dla przycisku pierwszego i drugiego będzą następujące:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
ColorDialog1->Color=Kolor1;
ColorDialog1->Execute();
Kolor1=ColorDialog1->Color;
RysujPasek(&Kolor1,&Kolor2);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
ColorDialog1->Color=Kolor2;
ColorDialog1->Execute();
Kolor2=ColorDialog1->Color;
RysujPasek(&Kolor1,&Kolor2);
}
```

Ostatnia linia wywołuje funkcję (właściwie będzie to metoda klasy `TForm1`), która będzie odpowiedzialna za narysowanie odpowiedniego paska, a którą musimy teraz napisać.

Najpierw zadeklarujmy odpowiednią metodę w sekcji Private klasy `TForm1`:

```
void RysujPasek(TColor*,TColor*);
```

A następnie na końcu pliku Unit1.cpp dopiszmy naszą metodę TForm1->RysujPasek(TColor*,TColor*)

```
void TForm1::RysujPasek(TColor* KolorGorny,TColor* KolorDolny)
{
int Max=Form1->ClientHeight;
for(int i=0;i<Max;i++)
            {
            unsigned int gradR=GetRValue(*KolorGorny)+
                  (GetRValue(*KolorDolny)-GetRValue(*KolorGorny))*i/Max;
            unsigned int gradG=GetGValue(*KolorGorny)+
                  (GetGValue(*KolorDolny)-GetGValue(*KolorGorny))*i/Max;
```

```
            unsigned int gradB=GetBValue(*KolorGorny)+
                  (GetBValue(*KolorDolny)-GetBValue(*KolorGorny))*i/Max;
            Canvas->MoveTo(0,i);
            Canvas->Pen->Color=(TColor)RGB(gradR,gradG,gradB);
            Canvas->LineTo(Form1->ClientWidth,i);
            }
}
```

Wywołanie tej metody powinno się znaleźć również w metodzie związanej z OnPaint, aby przypisane w kreatorze wartości zmiennych zgadzały się z początkowymi kolorami formy.

3. Dywan graficzny – wykorzystanie tablicy `Canvas->Pixels`

W nowej aplikacji, po dołączeniu biblioteki funkcji matematycznych poleceniem, w zależności od języka

```
#include <math.h>
```

lub

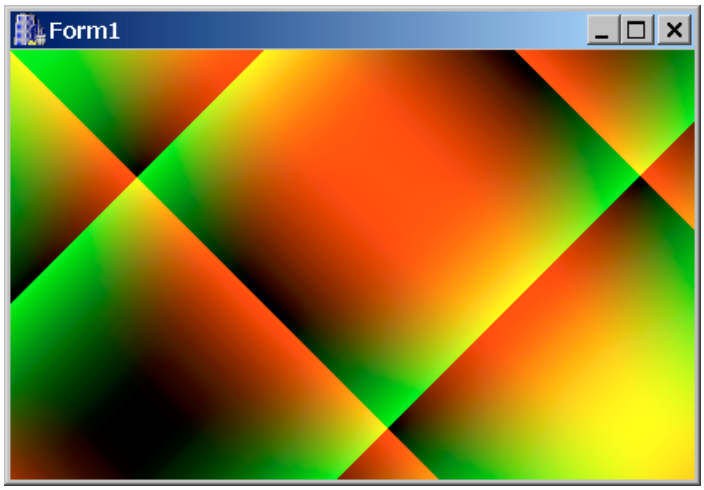
```
uses math;
```

w metodzie związanej z Form1->OnPaint wpisz następujące polecenia:

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
int r,g,b;
int x,y;
TColor Kolor;
for(x=0;x<ClientWidth;x++)
      for(y=0;y<ClientHeight;y++)
      {
      r=255*sin((x-y)/100.0);
      g=255*cos((x+y)/100.0);
      b=0;
      Kolor=(TColor)RGB(r,g,b);
      Canvas->Pixels[x][y]=Kolor;
      }
}
```

Wersja Delphi tej samej metody powinna wyglądać następująco:

```
procedure TForm1.FormPaint(Sender: TObject);
var   r,g,b :Integer;
      x,y :Integer;
      Kolor :TColor;
begin
for x:=0 to ClientWidth do
for y:=0 to ClientHeight do
    begin
    r:=Round(255*sin((x-y)/100));
    g:=Round(255*cos((x+y)/100));
    b:=0;
    Kolor:=RGB(r,g,b);
    Canvas.Pixels[x,y]:=Kolor;
    end;
end;
```



Metoda ta posługując się funkcjami trygonometrycznymi przypisuje kolor każdemu pikselowi po kolei.

Uwaga!
Piękno dywanu opiera się na oczywistym błędzie z punktu widzenia elegancji programowania. Wartości przypisywane do własności r i g obliczane są za pomocą funkcji sin(), która przyjmuje wartości z zakresu (-1, 1). A więc całe wyrażeni przyjmuje wartości naturalne od –255 do 255. Funkcja RGB przyjmuje trzy liczby typu BYTE (ośmiobitowe). Niejawne rzutowanie na ten typ nie oznacza wcale wzięcie wartości bezwzględnej, ponieważ ostatnie osiem bitów zostanie odczytane tak jak dla liczby dodatniej. Stąd na formie bieże się nagła zmiana koloru.

Funkcję obliczającą kolor można dowolnie modyfikować. Np. zmieniając komponent niebieski:

```
b=r+g;
```

Jeżeli chcemy uzyskać szum możemy wpisać:

```
r=random(255);
g=random(255);
b=random(255);
```

Uwaga!
W C++ Builder 3 i nowszych zamiast z tablicy Pixels należy korzystać ze znacznie szybszej własności `Picture->Bitmap->ScanLine`.

.

4. Negatyw (C++ Builder 3 lub nowszy)

Tworzymy aplikację z formą zawierającą obraz TImage (można ustawić własność Align = alClient). Wczytujemy obraz za pomocą edytora własności Picture. Można ustawić własność Stretch = true.

Własność ScanLine zwraca adres do pierwszego **bajtu** w linii obrazu (numer linii jest podawany jako indeks). Aby dowiedzieć się ile bajtów jest przeznaczone na każdą linię obrazu, a w konsekwencji ile bitów koduje kolor każdego punktu odejmujemy od siebie adresy dwóch sąsiednich linii:

```
int BytesPerScan = int(Image1->Picture->Bitmap->ScanLine[1])
                         - int(Image1->Picture->Bitmap->ScanLine[0]);
```

Tą informację można również uzyskać sprawdzając własność `Image1->Picture->Bitmap->PixelFormat`. Jeżeli użyte są kolory TrueColor, każdy punkt kodowany jest 24 bitami, a więc trzema bajtami.

ScanLine udostępnia jedynie adres pierwszego bajtu wybranej linii obrazu, a więc kolor niebieski. Odczytując kolejne bajty w pętli uzyskamy B1, G1, R1, B2, G2, R2, itd. „Odwracając" wartość (kolor = 255 – kolor) każdego koloru uzyskamy negatyw, a jest to tym łatwiejsze do realizacji, że można „odwracać" każdy bajt po kolei.

```
for (int y = 0; y<Image1->Picture->Bitmap->Height; y++)
        {
        Byte* p = (Byte*)Image1->Picture->Bitmap->ScanLine[y];
        //x nie indeksuje pixeli, a bajty
        for (int x = 1; x<BytesPerScan; x++) p[x]=255-p[x];
        }
```

Aby zobaczyć efekt na ekranie należy odświeżyć obiekt metodą `Image1->Refresh();`.

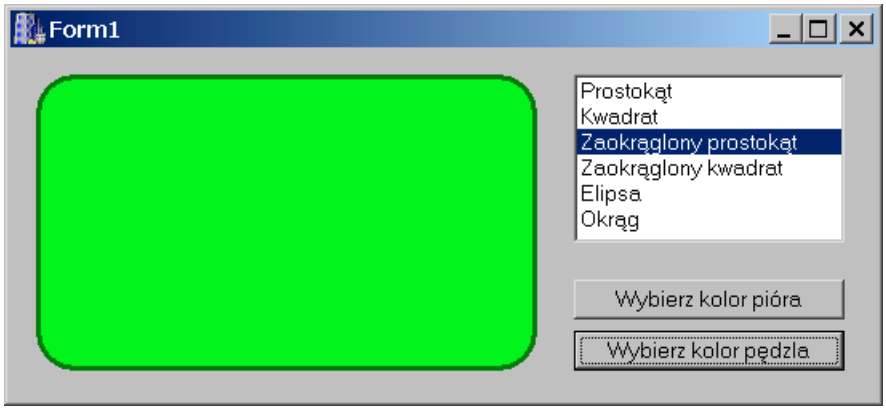
Uwaga! Taki sposób dostępu do kolorów nie zadziała jeżeli obraz nie jest kodowany za pomocą trójki liczb RGB (na przykład wówczas gdy jest to obraz z indeksowanymi kolorami).

Aby porównać szybkość dostępu do pixeli obrazu przy wykorzystaniu obu sposobów można zastąpić powyższą pętlę wykorzystującą ScanLine przez pętlę zmieniającą wartości w tablicy `Image1->Picture->Bitmap->Canvas->Pixels`:

```
for (int y = 0; y<Image1->Picture->Bitmap->Height; y++)
for (int x = 0; x<Image1->Picture->Bitmap->Width; x++)
//tu x indeksuje pixele
      {
      TColor kolor=Image1->Picture->Bitmap->Canvas->Pixels[x][y];
      Byte r=GetRValue(kolor);
      Byte g=GetGValue(kolor);
      Byte b=GetBValue(kolor);
      Image1->Picture->Bitmap->Canvas->Pixels[x][y]=RGB(255-r,255-g,255-b);
      }
```

5. TShape

Na formie umieścić obiekt klasy TShape oraz kontrolki podobnie jak na rysunku. Grubość pióra została zwiększona do 3 (`Shape->Pen->Width=3`).



Za pomocą listy będziemy kontrolować kształt TShape, a przyciskami zmieniać kolory pióra i pędzla (konturu i wypełnienia):

```
void __fastcall TForm1::ListBox1Click(TObject *Sender)
{
Shape1->Shape=ListBox1->ItemIndex;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
if (ColorDialog1->Execute()) Shape1->Pen->Color=ColorDialog1->Color;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
if (ColorDialog1->Execute()) Shape1->Brush->Color=ColorDialog1->Color;
}
```

V. Multimedia – TMediaPlayer

Celem tego ćwiczenia jest przygotowanie prostych odtwarzaczy plików audio i wideo. W każdym z przypadków na formie umieszczamy TMediaPlayer z palety System oraz komponent Panel z palety standardowej służący jako „ekran” w przypadku odtwarzacza plików wideo.

Komponent TMediaPlayer korzysta z systemowych bibliotek Window. Odtwarza nie tylko pliki typu wave, midi i avi, ale również wszystkie zarejestrowane typy kompresji (np. mpeg, w tym mp3). Informacje o koderach można znaleźć w Panelu sterowania\Multimedia\zakładka Urządzenia\Kodery-dekodery kompresujące audio i wideo.

1. Odtwarzacz wideo (AVI) – projektowanie RAD

1. W zależności od ustalenia wartości **Display** (ważne tylko w plikach wideo) plik zostanie odtworzony w nowo otwartym okienku (gdy Display ma wartość pustą) lub zostanie odtworzony we wskazanym przez użytkownika miejscu (rozwijalna lista pokazuje możliwości). Przypiszmy **MediaPlayer1->Display=Panel1**..
2. Należy teraz ustalić wartość **MediaPlayer1->FileName** wskazując na wybrany plik avi.
3. Uruchomienie MediaPlayer może odbyć się automatycznie, jeżeli własność **AutoOpen** jest włączona, lub za pomocą metody MediaPlayer->Open(). Metoda Open() otwiera plik, ale go nie odtwarza.
4. Można zażądać, aby MediaPlayer miał wyłączność na korzystanie z danego urządzenia ustalając **Sharable** na False.
5. Kontrola pliku, w tym jego odtworzenie, możliwe jest za pomocą przycisków MediaPlayer lub za pomocą odpowiadających im metod:
   MediaPlayer1->Play();
   MediaPlayer1->Stop();
   itd.
6. Film będzie odtwarzany w oryginalnych rozmiarach. Jeżeli chcemy go przeskalować, należy ustalić wartość nieupublicznionej (tzn. niedostępnej w Object Inspector) własności MediaPlayer1->DisplayRect wskazującą na prostokąt, w którym znajdować będzie się „ekran”. W naszym przykładzie niech to będzie **MediaPlayer1->DisplayRect=Panel1->ClientRect;**

Następne dwa przykłady przeczą idei programowania RAD, gdyż TMediaPlayer będzie tworzony i konfigurowany w całości dynamicznie w trakcie działania programu – jest to trudniejsze, ale konieczne np. przy pisaniu własnych obiektów i komponentów.

2. Odtwarzacz CD Audio – funkcja z dynamicznym wywoływaniem obiektu

Dodamy przycisk o nazwie btnStart, który będzie uruchamiał odtwarzanie płyty CD Audio. Całą obsługę CD-ROMu zamkniemy w funkcji CDAudioCommand(), która sama dynamicznie stworzy obiekt klasy TMediaPlayer. Ustalimy typ obsługiwanego urządzenia **DeviceType** na dtCDAudio (w większości przypadków, gdy odtwarzane są różne typy plików, najlepsza jest wartość domyślna dtAutoSelect automatycznie wykrywająca typ pliku na podstawie jego rozszerzenia) i zainicjujemy odtwarzanie muzyki. Ponieważ odtwarzanie płyt CD Audio jest kontrolowane sprzętowo, można usunąć obiekt z pamięci – funkcja służy jedynie do wysyłania odpowiednich instrukcji do odtwarzacza. W szczególności nawet po zamknięciu aplikacji płyta będzie nadal odtwarzana. Nie warto tworzyć dla CDAudio stale obecnego w pamięci obiektu, żeby jej nie marnować. Inaczej będzie w przypadku odtwarzaczy programowych, np. wav lub mp3.

Pisanie funkcji CDAudioCommand() zaczniemy od dodania do pliku nagłówkowego (Unit1.h) deklaracji odpowiedniej biblioteki

```
#include <vcl\mplayer.hpp>
```

W deklaracji klasy TForm1 w części private dodajemy deklarację metody i pomocniczego typu wyliczeniowego:

```
void __fastcall CDAudioCommand(int);
enum cdaCommands {Eject=-1, Stop, Play, Next} cdaCommand;
```

Następnie kopiujemy do zasadniczego pliku z implementacją klasy następujący kod (Unit1.cpp):

```
void __fastcall TForm1::CDAudioCommand(int command)
{
TMediaPlayer* CDAudio=new TMediaPlayer(Form1);
CDAudio->Visible=False;
CDAudio->Parent=Form1;
CDAudio->Shareable=True;
CDAudio->DeviceType=dtCDAudio;
CDAudio->Open();
switch (command)
      {
      case cdaStop: CDAudio->Stop(); break;
      case cdaPlay: CDAudio->Play(); break;
      case cdaNext: CDAudio->Next(); break;
      case cdaEject: CDAudio->Eject(); break;    }
      }
CDAudio->Close();
delete CDAudio;
}
```

Teraz do formy dodajemy przyciski, w których wywołujemy funkcję z jednym z argumentów

```
CDAudioCommand(Sender,cdaPlay);
CDAudioCommand(Sender,cdaStop);
CDAudioCommand(Sender,cdaNext);
CDAudioCommand(Sender,cdaEject);
```

Niestety, w sytuacji, w której poprzednio został uruchomiony inny program, który nie umożliwia współdzielenia sterownika urządzenia – pojawi się błąd.

Takie rozwiązanie zaoszczędza wykorzystywaną pamięć, ale uniemożliwia np. śledzenie pozycji w pliku (własności **TrackLength** i **Position**). Można więc zainicjować dynamicznie obiekt klasy TMediaPlayer w momencie tworzenia formy (tym razem wskaźnik powinien być zmienną dostępną dla całej klasy TForm1) i zniszczyć go dopiero w momencie zamykania aplikacji.

Zadanie
Funkcję CDAudioCommand wraz ze związanymi z nią deklaracjami umieścić w osobnym pliku, który będzie mógł być włączany do innych programów.

3. Odtwarzacz Wav/Mp3 – prawie komponent

Tu zastosujemy inną filozofię niż w poprzednich przykładach. Zainicjujemy komponent dostępny dla całej klasy TForm1 w pliku nagłówkowym

```
private:    // User declarations
      TMediaPlayer* Mp3Player; //zasadniczy odtwarzacz
      TTrackBar* Mp3PlayerTrackBar; //suwak pokazujący pozycję
      TTimer* Mp3PlayerTimer; //pozwoli kontrolować pozycję suwaka
      void __fastcall Mp3PlayerOnTimer(TObject*); //dwie metody zdarzeniowe
      void __fastcall Mp3PlayerPositionChange(TObject*);
```

i stworzymy go w kreatorze formy (tym razem posłużymy się oryginalnym panelem z przyciskami):

```
__fastcall TForm1::TForm1(TComponent* Owner)
```

```
      : TForm(Owner)
{
Mp3Player=new TMediaPlayer(Form1);
Mp3Player->Parent=Form1;
Mp3Player->Shareable=True;

//Wybór przycisków
Mplayer::TButtonSet Przyciski;
Przyciski << btStop << btPlay << btPause;
Mp3Player->VisibleButtons=Przyciski;

Mp3Player->Left=10;
Mp3Player->Width=Form1->ClientWidth-20;
Mp3Player->Top=10;
//Tu aż prosi się o wywołanie TOpenDialog
Mp3Player->FileName="g:\\Sultans Of Swing.mp3";
Form1->Caption=Mp3Player->FileName;
Mp3Player->Open();

Mp3PlayerTrackBar=new TTrackBar(Form1);
Mp3PlayerTrackBar->Left=Mp3Player->Left;
Mp3PlayerTrackBar->Top=Mp3Player->Top+Mp3Player->Height+10;
Mp3PlayerTrackBar->Width=Mp3Player->Width;
Mp3PlayerTrackBar->Frequency=Mp3Player->Length/10;
Mp3PlayerTrackBar->Max=Mp3Player->Length;
Mp3PlayerTrackBar->LineSize=10;
Mp3PlayerTrackBar->Parent=Form1;
Form1->ClientHeight=Mp3PlayerTrackBar->Top+Mp3PlayerTrackBar->Height+10;

Mp3PlayerTimer=new TTimer(Form1);
Mp3PlayerTimer->Interval=1000;

//Przypisanie zdarzeniom odpowiednich funkcji
//Zdarzenia są, jak widać, wskaźnikami, a więc przechowują adresy funkcji
//z którymi są skojarzone. Zwolnienie skojarzenia przez nadanie wartości 0
Mp3PlayerTimer->OnTimer=&Mp3PlayerOnTimer;
Mp3PlayerTrackBar->OnChange=&Mp3PlayerPositionChange;
}
```

Konieczne jest jeszcze zdefiniowanie dwóch funkcji związanych ze zdarzeniami:

```
void __fastcall TForm1::Mp3PlayerOnTimer(TObject* Sender)
{
Mp3PlayerTrackBar->Position=Mp3Player->Position;
}

void __fastcall TForm1::Mp3PlayerPositionChange(TObject* Sender)
{
boolean Playing=False;
if (Mp3Player->Mode==mpPlaying) Playing=True;
Mp3Player->Position=Mp3PlayerTrackBar->Position;
if (Playing) Mp3Player->Play();
}
```

Uwaga! Wszystkie, a przynajmniej ogromną większość tych czynności można by oczywiście wykonać znacznie łatwiej w trakcie projektowania formy. Jednak wiedza o tym jak przypisać funkcję do zdarzenia przyda się nam przy projektowaniu komponentów. Powyższe elementy wystarczyłoby umieścić nie bezpośrednio na formie, a na np. dynamicznie tworzonym panelu i zamknąć w osobnym obiekcie (teraz całość jest własnością i metodami TForm1), aby przeobrazić nasz odtwarzacz w komponent.

Uwaga! Aby ten przykład działał w systemie musi być zainstalowany codec MPEG Layer-3 (mp3). W przeciwnym przypadku należy zastąpić plik mp3 plikiem wav.

Elegancja programowania wymaga, aby w zdarzeniu OnClose formy umieścić instrukcję zwalniającą urządzenie wykorzystywane przez MediaPlayer

```
Mp3Player->Close();
delete Mp3Player;
```

choć oczywiście, zostanie ono i tak zwolnione przez system.

Zadanie
Zaprojektować aplikację, która o każdej pełnej godzinie będzie odtwarzała wybrany w trakcie projektowania plik audio.

VI. Klasy i komponenty w C++ Builder

1. Klasa TSFolders

Stworzymy klasę, która będzie posiadała kilka opublikowanych (published) własności typu łańcuchowego przechowujących nazwy katalogów: domyślnego katalogu dokumentów bieżącego użytkownika (Moje dokumenty), katalog systemowy Windows oraz katalog bieżący. Tylko ta ostatnia własność będzie mogła być zmieniana (do jej stworzenia skorzystamy z konstrukcji własności – __property).

Uwaga! Ściśle rzecz biorąc zmiennych obiektu (*data members*) nie powinno się nazywać własnościami. Słowo własności zarezerwowane jest zazwyczaj dla opisanej poniżej konstrukcji wykorzystującej słowo kluczowe __property będącej pomostem między polami i metodami.

Za pomocą menu File\New Application tworzymy projekt, w którym będziemy pisać i testować nową klasę. Następnie tworzymy plik (prawy klawisz myszy na zakładkach okna edycyjnego i pozycja menu kontekstowego 'Open File at Cursor') o nazwie SFolder (typ Unit).

Deklaracja klasy
W pliku nagłówkowym (Ctrl+F6) SFolder.h zapisujemy szkielet nowej klasy o nazwie TSFolder (ważne, aby tekst użytkownika znalazł się pomiędzy przygotowanymi przez C++ Buildera dyrektywami #define SFolderH i #endif:

```
class TSFolder
{
public:
TSFolder();
AnsiString MyDocumentsDir;
AnsiString WindowsDir;
AnsiString WindowsSystemDir;
};
```

Trzy zadeklarowane pola obiektu TSFolder – zmienne typu AnsiString będą ustalane w konstruktorze klasy TSFolder() i nie będzie w trakcie bieżącej sesji Windows potrzeby, aby ich zawartość odświeżać. Informacje o odpowiednich katalogach przechowywane są w rejestrach, a o niektórych z nich można również dowiedzieć się korzystając z funkcji systemowych WinAPI.

Zajmijmy się teraz pisaniem konstruktora naszej klasy. Konstruktor w C++ musi być metodą o nazwie identycznej z nazwą klasy (konstruktor może być przeciążany). Przejdźmy do pliku SFolder.cpp (Ctrl+F6) i na końcu pliku dopiszmy:

```
TSFolder::TSFolder()
{
char tmpdir[MAX_PATH];
};
```

W tej chwili konstruktor powołuje jedynie zmienną łańcuchową, która będzie przechowywała chwilowo kolejne odczytywane nazwy katalogów. Jest to konieczne ze względu na brak zgodności funkcji API i niektórych klas VCL ze zmienną typu AnsiString. Długość łańcucha jest równa predefiniowanej stałej MAX_PATH przechowującej maksymalną dopuszczalną długość ścieżki w systemie.

Funkcje WinAPI GetWindowsDirectory i GetSystemDirectory
Nazwy katalogu Windows i katalogu systemowego Windows można odczytać za pomocą funkcji WinAPI GetWindowsDirectory oraz GetSystemDirectory (więcej informacji można znaleźć w MS Help dołączanym do Delphi i C++ Buildera). Ich składnia jest jednakowa: pierwszym argumentem jest adres (wskaźnik) łańcucha, a drugim maksymalna długość przechowywanej w nim nazwy. W naszym przypadku są to odpowiednio tmpdir i MAX_PATH.

Uzupełnijmy konstruktor o wywołania obu funkcji:

```
//WindowsDir
GetWindowsDirectory(tmpdir,MAX_PATH);
WindowsDir=(AnsiString)tmpdir;

//WindowsSystemDir
GetSystemDirectory(tmpdir,MAX_PATH);
WindowsSystemDir=(AnsiString)tmpdir;
```

Aby przetestować działanie tych funkcji możemy do formy projektu, który zamierzamy użyć do testowania naszej klasy dodać dwa obiekty TLabel i do konstruktora formy dopisać jeden z wariantów stworzenia obiektu klasy TSFolder (związek między obiektem i klasą jest taki sam jak między zmienną i typem zmiennej):

```
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
/*
TSFolder SFolder;
Label1->Caption=SFolder.WindowsDir;
Label2->Caption=SFolder.WindowsSystemDir;
*/

TSFolder* SFolder=new TSFolder();
Label1->Caption=SFolder->WindowsDir;
Label2->Caption=SFolder->WindowsSystemDir;
//delete SFolder;
}
```

Uwaga! Aby aplikacja widziała stworzoną przez nas klasę, trzeba jej wskazać pliki, w których ją zapisaliśmy, a więc dodać np. w jej pliku nagłówkowym komendę `#include „SFolder.cpp”`.

Pierwszy statyczny sposób, zakomentowany – nie powinien być używany ze względu na sposób korzystania z pamięci. W ten sposób nie można powoływać komponentów VCL. Sposób drugi (dynamiczny) korzystający ze słowa kluczowego **new** jest zalecanym sposobem tworzenia obiektów w C++ Builderze. Jego zaletą jest możliwość zwolnienia pamięci używanej przez niepotrzebny już obiekt za pomocą słowa kluczowego **delete**. Jeżeli chcemy, aby stworzony w konstruktorze formy obiekt SFolder klasy TSFolder był dostępny w całej aplikacji powinniśmy w deklaracji klasy obiektu umieścić, najlepiej prywatny, wskaźnik do naszej klasy (tj. dopisać linię `TSFolder* SFolder;`), a w konstruktorze powołać odpowiedni obiekt przypisując tej zmiennej adres do niego (tzn. `SFolder=new TSFolder();`)

Odczytywanie z rejestru nazwy katalogu „Moje dokumenty”
Kolejną nazwę katalogów można odczytać jedynie z rejestru systemowego (od Windows 95 SE). Katalog, którego nazwę chcemy poznać należy do grupy tzw. katalogów powłoki systemu (shell folders), do której należą również katalog pulpitu, menu Start, menu Start\Programy, katalog Autostartu, otoczenia sieciowego itd. Nazwy tych katalogów dla bieżącego użytkownika znajdują się w kluczu rejestru:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders

Można to sprawdzić uruchamiając edytor rejestru (polecenie **regedit** z linii komend). Warto zwrócić od razu uwagę na podobny klucz User Shell Folders.
Chcąc odczytać interesującą nas nazwę musimy odczytać wartość z `Personal` (w systemach rodziny Win 9*, w których jest tylko jeden użytkownik najczęściej jest to „C:\Moje dokumenty”). Do czytania i pisania w rejestrze systemu służy klasa VCL TRegistry zadeklarowana w pliku vcl/registry.hpp (należy dodać go w naszym pliku nagłówkowym klasy poleceniem `#include <vcl/registry.hpp>`). Następnie do konstruktora klasy należy dopisać odpowiednie wywołanie obiektu typu TRegistry:

```
//MyDocumentsDir
TRegistry* Registry=new TRegistry;
Registry->RootKey = HKEY_CURRENT_USER;
```

```
AnsiString Key =
"\\Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell Folders";
if(Registry->KeyExists(Key))
      {
      Registry->OpenKey(Key,false);
      if (Registry->ValueExists("Personal"))
            {
            MyDocumentsDir=Registry->ReadString("Personal");
            }
      }
Registry->CloseKey();
delete Registry;
```

Uwaga! Ze względu na konwencję przyjętą w C++ znaki slash „\” muszą być zapisywane za pomocą podwójnego znaku „\\”.

Ze względu na przejrzystość kodu pomijamy wszelką obsługę błędów.
Do kodu testującego naszą klasę można teraz dodać (po dodaniu do formy jeszcze jednego Labela) przypisanie `Label3->Caption=SFolder->MyDocumentsDir;`. Po uruchomieniu programu powinniśmy zobaczyć nazwy trzech interesujących nas katalogów.

Własność CurrentDir (_ _ property)
Możliwość definiowania „aktywnych własności” za pomocą słowa kluczowego _ _ property pojawiła się dopiero w kompilatorach firmy Borland od wersji 5, m.in. w C++ Builderze. Nie należy do standardu języka C++, ale jest niezwykle wygodnym narzędziem.

Do deklaracji klasy TSFolders (w pliku nagłówkowym) dopiszmy następującą linię w sekcji public:

```
__property AnsiString WorkingDir={read=GetWorkingDir,
                                  write=ChangeWorkingDir};
```

Oznacza ona, że przy próbie odczytania wartości własności WorkingDir będzie uruchomiona metoda ReadWorkingDir, której wartość powinna być typu zadeklarowanej własności (tj. AnsiString), a próba zmiany wartości tej własności uruchomi metodę ChangeWorkingDir, której argumentem jest AnsiString. Musimy zatem zadeklarować i zaimplementować odpowiednie metody. Do sekcji private dopiszmy:

```
private:
AnsiString GetWorkingDir();
void ChangeWorkingDir(AnsiString AWorkingDir);
```

a w pliku SFolder.cpp zdefiniujmy następująco ciała tych metod:

```
AnsiString TSFolder::GetWorkingDir()
{
char tmpdir[MAX_PATH];
GetCurrentDirectory(MAX_PATH,tmpdir);
return (AnsiString)tmpdir;
};

void TSFolder::ChangeWorkingDir(AnsiString AWorkingDir)
{
SetCurrentDirectory(AWorkingDir.c_str());
};
```

W tej chwili, jeżeli użyjemy przypisania do lub czytania z własności `SFolder->WorkingDir`, wykonane zostaną odpowiednie funkcje WinAPI Get/SetCurrentDirectory do odczytania lub zmiany bieżącego katalogu. Aby przetestować działanie tego fragmentu kodu możesz dodać TEdit lub TOpenDialog do zmiany katalogu i kolejny Label do jego pokazania.

Zadanie:
Przeciążyć konstruktor tak, aby mógł przyjmować jako argument nazwę nowego bieżącego katalogu.

2. Klasa TStoper

Bardzo łatwo stworzyć prosty stoper korzystając z komponentu TButton jako klasy bazowej.

Deklarujemy klasę TStoper:

```
class TStoper : public TButton
{
public:
_fastcall TStoper(TComponent* Owner);
private:
long miliseconds;
TTimer* Timer;
void __fastcall Start(TObject* Sender);
void __fastcall Stop(TObject* Sender);
void __fastcall Reset(TObject* Sender);
void __fastcall Tick(TObject* Sender);
};
```

zawierającą konstruktor typowy dla komponentów (argument, którym przekazuje się właściciela obiektu – nie mylić z rodzicem). W sekcji prywatnej umieszczona jest deklaracja zmiennej miliseconds, która będzie przechowywała ilość tysięcznych sekundy od naciśnięcia przycisku (TStoper dziedziczy z TButton, który jest po prostu przyciskiem), wkaźnik do TTimera („impulsowego zegara") oraz czterech metod, których argumenty są tak dobrane, żeby odpowiadały zdarzeniom OnClick w TButton i OnTimer w TTimer.

W konstruktorze przypisujemy własności geometryczne (domyślne, użytkownik może je zmienić) komponentu, napis na przycisku.

```
_fastcall TStoper::TStoper(TComponent* Owner):TButton(Owner)
{
Width=200;
Height=30;
Caption="Start";
miliseconds=0;

Timer=new TTimer(this);
Timer->Interval=1;
Timer->Enabled=false;
Timer->OnTimer=&Tick;

OnClick=&Start;
};
```

Najważniejsze jest jednak stworzenie obiektu typu TTimer (jego adres przechowuje zadeklarowany w nagłówku wskaźnik Timer), ustalenie czasu między impulsami na jedną milisekundę oraz przypisanie zdarzeniu OnTimer metody Tick:

```
Timer->OnTimer=&Tick;
```

Widać, że przypisanie metody do zdarzenia polega na ustaleniu wartości odpowiednich wskaźników, którymi po prostu są zdarzenia, równej adresowi metod, z którymi mają być związane. Nie ma istotnej różnicy między zapisem `OnClick=&Start`**, a** `OnClick=Start`**.**

Metoda ta ma zwiększać naliczoną ilość milisekund oraz wyświetlać je na przycisku:

```
void __fastcall TStoper::Tick(TObject* Sender)
{
miliseconds++;
Caption=AnsiString((double)miliseconds)+" ms";
};
```

Ostatnim poleceniem konstruktora jest przypisanie zdarzeniu OnClick przycisku-stopera metody Start:

```
void __fastcall TStoper::Start(TObject* Sender)
{
Timer->Enabled=true;
OnClick=&Stop;
};
```

która uruchamia Timer i zmienia metodę zdarzeniową skojarzoną z OnClick na Stop.

Z kolei Stop:

```
void __fastcall TStoper::Stop(TObject* Sender)
{
Timer->Enabled=false;
Caption="Reset ("+Caption+")";
OnClick=&Reset;
};
```

Wyświetla ostateczną ilość naliczonych milisekund i umożliwia wyzerowanie zegara zmieniając jeszcze raz metodę zdarzeniową na Reset, która pozwala na ponowne uruchomienie Stopera metodą Start:

```
void __fastcall TStoper::Reset(TObject* Sender)
{
Caption="Start";
miliseconds=0;
OnClick=&Start;
};
```

Zadanie
Zmienić format wyświetlania czasu na hh:mm:ss:ms dodając własności hours, minutes, seconds do grupy private.

3. Komponenty TLink i TMail

Należy pobrać z sieci pliki link.h i link.cpp lub stworzyć je za pomocą notatnika kopiując ich zawartość:

Plik link.h

```
//---------------------------------------------------------------------------
#ifndef LinkH
#define LinkH
//---------------------------------------------------------------------------
#include <vcl\SysUtils.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Forms.hpp>
#include <vcl\StdCtrls.hpp>
//---------------------------------------------------------------------------

const Controls::TCursor crLink=1;
const Controls::TCursor crMail=2;

class TLink_base : public TLabel
{
private:
protected:
      void __fastcall ColorBlack(TObject* Sender,TmouseButton
                        Button,Classes::TShiftState Shift,int X,int Y);
```

```
      void __fastcall ColorNavy(TObject* Sender,TMouseButton
                        Button,Classes::TShiftState Shift,int X,int Y);
      AnsiString FAddress;
      virtual void __fastcall CheckAddress(System::AnsiString)=0;
public:
      __fastcall TLink_base(TComponent* Owner);
      void __fastcall Connect(TObject* Sender);
__published:
      __property System::AnsiString Address = {read=FAddress,
                        write=CheckAddress, nodefault};
};

class TLink : public TLink_base
{
private:
      void __fastcall CheckAddress(System::AnsiString);
public:
      __fastcall TLink(TComponent* Owner);
};

//---------------------------------------------------------------------------
class TMail : public TLink_base
{
private:
      void __fastcall CheckAddress(System::AnsiString);
protected:
public:
      __fastcall TMail(TComponent* Owner);
__published:
};

//---------------------------------------------------------------------------
#endif
```

Plik link.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "Link.h"
#include <shellapi.h>
//---------------------------------------------------------------------------
static inline TLink *ValidCtrCheck()
{
      return new TLink(NULL);
}
//---------------------------------------------------------------------------
__fastcall TLink_base::TLink_base(TComponent* Owner)
      : TLabel(Owner)
{
//Set<TFontStyle, fsBold, fsStrikeOut>  LinkStyle;
TFontStyles LinkStyle;
LinkStyle << fsUnderline;
Font->Color=clNavy;
Font->Style=LinkStyle;
OnClick=Connect;
OnMouseDown=ColorBlack;
OnMouseUp=ColorNavy;
Screen->Cursors[crLink] = LoadCursor(HInstance,"LINK");
Cursor=crLink;
}
```

```
void __fastcall TLink_base::Connect(TObject* Sender)
      {ShellExecute(NULL,"open",Address.c_str(),"","",SW_NORMAL);}

void __fastcall TLink_base::ColorBlack(TObject* Sender,TMouseButton
Button,Classes::TShiftState Shift,int X,int Y)
      {Font->Color=clBlack;}

void __fastcall TLink_base::ColorNavy(TObject* Sender,TMouseButton
Button,Classes::TShiftState Shift,int X,int Y)
      {Font->Color=clNavy;}

//---------------------------------------------------------------------------

__fastcall TLink::TLink(TComponent* Owner)
      : TLink_base(Owner)
{
Caption="http://www.phys.uni.torun.pl/~jacek";
FAddress=Caption;
Hint="Link to "+Caption;
ShowHint=true;
}

void __fastcall TLink::CheckAddress(System::AnsiString _URL)
      {
    if (_URL.SubString(1,7)!="http://") _URL="http://"+_URL;
    FAddress=_URL;
    }

//---------------------------------------------------------------------------
__fastcall TMail::TMail(TComponent* Owner)
      : TLink_base(Owner)
{
Caption="jacek@phys.uni.torun.pl";
FAddress="mailto:"+Caption;
Hint="Send a message to "+Caption;
ShowHint=true;
}


void __fastcall TMail::CheckAddress(System::AnsiString _Address)
      {
      if (_Address.SubString(1,7)!="mailto:")
            {_Address="mailto:"+_Address; FAddress=_Address;}

      int hash=0;
      for(int i=0;i<_Address.Length();i++)
            if ((_Address.c_str())[i]=='@') hash=i;
      if (hash==0) {ShowMessage("E-mail adress should include '@'.");} else
            {FAddress=_Address;};
    }
//---------------------------------------------------------------------------
namespace Link
{
      void __fastcall Register()
      {
      TComponentClass classes[2] = {__classid(TLink), __classid(TMail)};
            RegisterComponents("JM", classes, 1);
      }
}
//---------------------------------------------------------------------------
```

Metody zdarzeniowe
W pliku nagłówkowym zadeklarowano trzy obiekty: TLink_base dziedziczący ze standardowej klasy VCL TLabel i dwie dziedziczące po nim klasy TLink i TMail.

Uwaga! Warto zauważyć, że konstruktory każdego komponentu wywołują konstruktory klas bazowych przed wykonaniem własnych specyficznych instrukcji.

Zadeklarowane w TLink_base metody ColorBlack i ColorNavy mają takie argumenty i wartość, aby mogły być metodami zdarzeniowymi zdarzeń OnMouseDown i OnMouseUp (tzn. mają identyczną postać jak funkcje tworzone przez środowisko dla tych zdarzeń dla TLabel). Podobnie metoda Connect jest dopasowana do zdarzenia OnClick. Przypisanie do odpowiednich zdarzeń znajduje się w konstruktorze TLink_base:

```
OnClick=Connect;
OnMouseDown=ColorBlack;
OnMouseUp=ColorNavy;
```

Zadania tych metod są bardzo proste. Connect wywołuje funkcję ShellExecute (akcja systemu związana z rozszerzeniem pliku lub protokołem argumentu, w obiektach potomnych będą to protokoły http: i mailto:). Identyczną reakcję systemu można uzyskać korzystając z polecenia start w linii komend. Funkcje ColorBlack i ColorNavy zmieniają kolor napisu.

Metoda wirtualna / Klasa abstrakcyjna
W klasie bazowej (TLink_base) zadeklarowana jest metoda wirtualna CheckAdress. Słowo kluczowe virtual powoduje, że w klasie potomnej wywoływana jest odpowiednia metoda klasy potomnej, nawet jeżeli obiekt został powołany ze wskaźnikiem klasy bazowej. Przypisanie tej metodzie zera powoduje, że metoda staje się pure virtual i w ten sposób klasa staje się klasą abstrakcyjną – nie można deklarować obiektów tej klasy.

W klasach potomnych metody CheckAdress sprawdzają poprawność zmiennej prywatnej Faddress, w której przechowywany jest adres URL (w TLink) lub adresu e-mail (w TEmail). W pierwszym przypadku, w razie potrzeby, dodawany jest przedrostek „http://”, w drugim dodawana jest nazwa protokołu „mailto:” i sprawdzana jest obecność „małpy” @.

Format napisu
W konstruktorze klasy TLink_base ustala się wygląd komponentu (podkreślenie, kolor, itp.) następującymi poleceniami:

```
TFontStyles LinkStyle;
LinkStyle << fsUnderline;
Font->Color=clNavy;
Font->Style=LinkStyle;
```

W pierwszej linii zostaje zdefiniowany zbiór pusty typu TfontStyles określający wygląd czcionki (możliwe elementy tego zbioru wyliczone są w TFontStyle). Do tego zbioru zostaje włożony jedynie fsUnderline, tzn., że od normalnego wyglądu napis będzie różnił się jedynie podkreśleniem.

Dodatkowe kursory
Za pomocą narzędzia Image Editor (menu Tools) można stworzyć kursor (zapisany został pod nazwą LINK. Kursor można zarejestrować poleceniem
`Screen->Cursors[crLink]=LoadCursor(HInstance,"LINK");`
i przypisać go zmiennej Cursor odziedziczonej z TLabel (z wartościami większymi od zera). To powinno spowodować zmianę kursora przy najechaniu na komponent.

W C++ Builder 5 można znaleźć charakterystyczne dla linków „łapki” wśród kursorów gotowych (o numerach mniejszych od zera) i nie trzeba tworzyć odpowiedniego wzoru samodzielnie.

Własność Address
Najważniejszą nowością w tych komponentach jest opublikowana własność Address, którą można edytować za pomocą inspektora obiektów. Wpisując poprawny adres e-mail w TMail (np. login@serwer.domena.pl) i adres WWW w TLink (np. www.domena.pl/~login) możemy sprawdzić działanie komponentów.

Rejestracja komponentu
Najbardziej podatny na błędy jest etap rejestrowania komponentu w środowisku Delphi / C++ Builder. W tym celu do kodu należy dopisać:

```
static inline TLink *ValidCtrCheck()
{
      return new TLink(NULL);
}
```

oraz

```
namespace Link
{
void __fastcall Register()
      {
      TComponentClass classes[2] = {__classid(TLink), __classid(TMail)};
      RegisterComponents("JM", classes, 1);
      }
}
```

Uwaga! Namespace musi mieć pierwszą literę dużą i nie może mieć dużych liter w środku.

Uwaga 2!
1) W nowszych wersjach C++ Buildera (np. 5) funkcja Register musi być poprzedzona makrem PACKAGE (odpowiada za współpracę z systemem pakietów – pliki .BPL). PACKAGE należy też dodać do każdej deklaracji klasy-komponentu (class PACKAGE Tlink).
2) Kolejną nowością jest konieczność zadeklarowania modułu jako pakietu za pomocą dyrektywy `#pragma package(smart_init)` (zazwyczaj w pliku cpp pod dyrektywą włączenia pliku nagłówkowego – pod deklaracją klasy).
3) Nowsze wersje zawierają również mechanizm umożliwiający upewnienie się przed instacją komponentu, że jego klasa nie jest klasą abstrakcyjną (nie zawiera metod pure virtual):

```
static inline void ValidCtrCheck(TLink *)
{
        new TLink(NULL);
}

static inline void ValidCtrCheck(TMail *)
{
        new TMail(NULL);
}
```

4) Następną rzeczą jest zastąpienie pliku .res zawierającego bitmapy reprezentujące komponenty na pliki **.dcr** (identyczne jak w Delphi)
5) Ostatnią rzeczą na którą warto zwrócić uwagę jest Package Collection Editor znajdujący się w Menu Tools.

Uwaga! W razie niepowodzenia w rejestracji komponentu (np. zniknięcie wszystkich komponentów VCL i/lub błąd przy uruchomieniu Buildera) należy w katalogu BIN podmienić wadliwy plik cmplib32.ccl przez jego backup cmplib32.~cc.

Ikony komponentów muszą znaleźć się w pliku **link.res** (ich nazwy muszą być identyczne z nazwami klas)

4. Komponenty TLink i TMail – drobne modyfikacje

Stworzony przez nas komponent TLink można na kilka sposobów poprawić:
1) należy usunąć własności (np. Caption, Color i kilka innych) i zdarzenia (wszystkie) do których programista nie powinien mieć dostępu.
2) Dodać własności (poza obecnym już Address) Description, ActiveColor, InactiveColor.

W przeciwieństwie do Delphi w C++, a w szczególności w C++ Builder, istnieje możliwość dziedziczenia prywatnego (w deklaracji klasy potomnej przed wskazaniem klasy bazowej należy napisać `private` lub `protected`). W naszej sytuacji nie jest to jednak wygodne wyjście – prywatne, i w konsekwencji niedostępne, staną się wszystkie własności i metody bazowego komponentu. Także konstruktor. Wygodniej jest skorzystać z możliwości odwrotnej – upublicznienia własności i zdarzeń. Dzięki temu, że programiści biblioteki VCL dla większości komponentów stworzyli obiekty pośrednie (zawsze z Custom w nazwie) np. TCustomLabel, które posiadają niemal pełną funkcjonalność umieszczonych na paletach komponentów obiektów z wyjątkiem tego, że ich metody i własności zadeklarowane są w sekcji `protected`.
W tej sytuacji, aby ukryć zbędne metody i zdarzenia musimy zmienić klasę bazową na `TCustomLabel`:

```
class TLink_base : public TCustomLabel
{
```

Oznacza to również modyfikację wywołania konstruktora klasy bazowej w naszym konstruktorze:

```
__fastcall TLink_base::TLink_base(TComponent* Owner)
      : TCustomLabel(Owner)
{
```

W tej chwili po zarejestrowaniu obiekt nie posiadałby większości własności (poza odziedziczonymi z TControl) i wszystkich zdarzeń. Część z nich znowu możemy udostępnić dodając do sekcji __published:

```
__published:
    __property Align;
    __property Enabled;
    __property Font;
    __property Transparent;
    __property Visible;
```

Pozostałe pozostawimy w ukryciu ustalając ich wartość w konstruktorze:

```
ShowAccelChar=false;
WordWrap=false;
```

Ad 2) Dodamy również kolejne własne własności:

```
protected:
    AnsiString FDescription;
    TColor FInactiveColor;
    TColor FActiveColor;

__published:
    __property System::AnsiString Description =
                {read=FDescription, write=SetDescription, nodefault};
    __property TColor InactiveColor =
                {read=FInactiveColor, write=SetColor, nodefault};
    __property TColor ActiveColor =
                {read=FActiveColor, write=FActiveColor, nodefault};
```

`InactiveColor` i `ActiveColor` powinny być aktywowane w konstruktorze:

```
FInactiveColor=clNavy;
FActiveColor=clBlack;
```

Metody `SetDescription()` i `SetColor()` powinny wyglądać następująco:

```
void __fastcall TLink_base::SetColor(TColor AColor)
      {
      FInactiveColor=AColor;
      Font->Color=FInactiveColor;
      }

void __fastcall TLink_base::SetDescription(System::AnsiString ADescription)
      {
      FDescription=ADescription;
      Caption=FDescription;
      }
```

Należy również zastąpić metody `ColorNavy()` i `ColorBlack()` przez

```
void __fastcall TLink_base::ChangeColorActive(TObject* Sender,
            TMouseButton Button,Classes::TShiftState Shift,int X,int Y)
      {Font->Color=FActiveColor;}

void __fastcall TLink_base::ChangeColorInactive(TObject* Sender,
            TMouseButton Button,Classes::TShiftState Shift,int X,int Y)
      {Font->Color=FInactiveColor;}
```

Uwaga!
Istotną rzeczą jest taka modyfikacja konstruktorów klas potomnych `TLink` i `TMail`, aby nie odpwoływały się do własności `Caption`, a zamiast tego, żeby ustalały własność `Description`.

5. Zdarzenia komponentu TStoper (zadanie)

Należy przygotować komponent `TStoper`. Poza funkcją rejestrującą należy dodać do niego zdarzenia `OnStart`, `OnStop` i `OnReset`. Zdarzenia deklaruje się analogicznie jak własności, z tym, że nie odwołują się do metod komponentu, a do prywatnego wskaźnika do metody zdarzeniowej (`TNotifyEvent` jest takim wskaźnikiem), z którą może ją powiązać programista. Innymi słowi zdarzenie jest formą przekazywania wskaźnika metody do obiektu.

```
//Zdarzenia
private:
      TNotifyEvent FOnStart;
      TNotifyEvent FOnStop;
      TNotifyEvent FOnReset;
__published:
      __property TNotifyEvent OnStart = {read=FOnStart, write=FOnStart};
      __property TNotifyEvent OnStop = {read=FOnStop, write=FOnStop};
      __property TNotifyEvent OnReset = {read=FOnReset, write=FOnReset};
```

Należy jednak uważać przy wywoływaniu wskaźników z komponentu ponieważ do zdarzenie nie musi być nic przypisane i w takiej sytuacji pojawiłby się błąd. Zatem przed uruchomieniem metody np. `FOnStart()` należy sprawdzić czy `FOnStart` nie ma przypadkiem wartości `NULL`.

```
void __fastcall TStoper::Start(TObject* Sender)
{
Timer->Enabled=true;
OnClick=&Stop;
if (FOnStart!=NULL) FOnStart(this);
};
```

VII. Klasy i komponenty w Delphi

1. Klasa TSFolders

Stworzymy klasę, która będzie posiadała kilka opublikowanych (published) własności typu łańcuchowego przechowujących nazwy katalogów: domyślnego katalogu dokumentów bieżącego użytkownika (Moje dokumenty), katalog systemowy Windows oraz katalog bieżący. Tylko ta ostatnia własność będzie mogła być zmieniana (do jej stworzenia skorzystamy z konstrukcji własności – property).

Uwaga! Ściśle rzecz biorąc w Pascalu zmiennych obiektu (w C++ nazywa się je *data members*) nie powinno się nazywać własnościami, a polami (*fields*). Słowo własności zarezerwowane jest zazwyczaj dla opisanej poniżej konstrukcji wykorzystującej słowo kluczowe property będącej pomostem między polami i metodami.

Za pomocą menu File\New Application tworzymy projekt, w którym będziemy pisać i testować nową klasę. Następnie tworzymy plik (prawy klawisz myszy na zakładkach okna edycyjnego i pozycja menu kontekstowego 'Open File at Cursor') o nazwie SFolder (typ Unit).

Deklaracja klasy
W sekcji interface zapisujemy szkielet nowej klasy o nazwie TSFolder:

```
type
    TSFolder = class
    public
          MyDocumentsDir :string;
          WindowsDir :string;
          WindowsSystemDir :string;
          constructor Create;
    end;
```

Trzy zadeklarowane pola obiektu TSFolder – zmienne typu AnsiString będą ustalane w konstruktorze klasy TSFolder.Create i nie będzie w trakcie bieżącej sesji Windows potrzeby, aby ich zawartość odświeżać. Informacje o odpowiednich katalogach przechowywane są w rejestrach, a o niektórych z nich można również dowiedzieć się korzystając z funkcji systemowych WinAPI.

Zajmijmy się teraz pisaniem konstruktora naszej klasy. Konstruktor w Object Pascalu może mieć dowolną nazwę (w odróżnieniu od C++ w deklaracji klasy wyróżnia go słowo kluczowe `constructor`), jednak zwyczajowo w Delphi używa się nazwy Create (tak nazywają się wszystkie konstruktory klas VCL). W sekcji implementation zapiszmy konstruktor:

```
constructor TSFolder.Create;
var tmpdir :array[0..MAX_PATH] of char;
begin

end;
```

W tej chwili konstruktor powołuje jedynie zmienną łańcuchową, która będzie przechowywała chwilowo kolejne odczytywane nazwy katalogów. Jest to konieczne ze względu na brak zgodności funkcji API i niektórych klas VCL ze zmienną typu AnsiString. Długość łańcucha jest równa predefiniowanej stałej MAX_PATH przechowującej maksymalną dopuszczalną długość ścieżki w systemie. Uwaga! Aby ta stała była widoczna należy zadeklarować uzycie modułu Windows (`uses Windows;`) na szczycie w sekcji interface.

Funkcje WinAPI GetWindowsDirectory i GetSystemDirectory
Nazwy katalogu Windows i katalogu systemowego Windows można odczytać za pomocą funkcji WinAPI GetWindowsDirectory oraz GetSystemDirectory (więcej informacji można znaleźć w MS Help dołączanym do Delphi i C++ Buildera). Ich składnia jest jednakowa: pierwszym argumentem jest adres (wskaźnik) łańcucha, a drugim maksymalna długość przechowywanej w nim nazwy. W naszym przypadku są to odpowiednio tmpdir i MAX_PATH. (Podobnie jak stała MAX_PATH funkcje te wymagają zadeklarowania `uses Windows;`)

Uzupełnijmy konstruktor o wywołania obu funkcji:

```
//WindowsDir
GetWindowsDirectory(tmpdir,MAX_PATH);
WindowsDir:=string(tmpdir);

//WindowsSystemDir
GetSystemDirectory(tmpdir,MAX_PATH);
WindowsSystemDir:=string(tmpdir);
```

Aby przetestować działanie tych funkcji możemy do formy projektu, który zamierzamy używać do testowania naszej klasy dodać dwa obiekty TLabel i do jej metody zdarzeniowej OnShow powołanie obiektu klasy TSFolder (związek między obiektem i klasą jest taki sam jak między zmienną i typem zmiennej):

```
procedure TForm1.FormCreate(Sender: TObject);
var SFolder1 :TSFolder;
begin
SFolder1:=TSFolder.Create;
Label1.Caption:=SFolder1.WindowsDir;
Label2.Caption:=SFolder1.WindowsSystemDir;
end;
```

Uwaga! Aby aplikacja widziała stworzoną przez nas klasę, trzeba jej wskazać pliki, w których ją zapisaliśmy, a więc dodać w sekcji interface do uses moduł `SFolder`.

W przeciwieństwie do Turbo Pascala, w konstrukcji `var SFolder1 :TSFolder;` SFolder1 jest tylko wskaźnikiem typu klasy TSFolder i musi być jeszcze zainicjowany przez wywołanie jego konstruktora `SFolder1:=TSFolder.Create;`.

Jeżeli chcemy, aby stworzony w konstruktorze formy obiekt SFolder klasy TSFolder był dostępny w całej aplikacji powinniśmy deklarację obiektu `SFolder1 :TSFolder;` umieścić w deklaracji klasy TFform1, najlepiej w jej sekcji private;

Odczytywanie z rejestru nazwy katalogu „Moje dokumenty”
Kolejną nazwę katalogów można odczytać jedynie z rejestru systemowego (od Windows 95 SE). Katalog, którego nazwę chcemy poznać należy do grupy tzw. katalogów powłoki systemu (shell folders), do której należą również katalog pulpitu, menu Start, menu Start\Programy, katalog Autostartu, otoczenia sieciowego itd. Nazwy tych katalogów dla bieżącego użytkownika znajdują się w kluczu rejestru:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders

Można to sprawdzić uruchamiając edytor rejestru (polecenie **regedit** z linii komend). Warto zwrócić od razu uwagę na podobny klucz User Shell Folders.
Chcąc odczytać interesującą nas nazwę musimy odczytać wartość z `Personal` (w systemach rodziny Win 9*, w których jest tylko jeden użytkownik najczęściej jest to „C:\Moje dokumenty”). Do czytania i pisania w rejestrze systemu służy klasa VCL TRegistry (należy dodać do wykorzystywanych modułów Registry).
Następnie do konstruktora klasy należy dopisać odpowiednie wywołanie obiektu typu TRegistry:

```
//MyDocumentsDir
Registry:=TRegistry.Create;
Registry.RootKey:=HKEY_CURRENT_USER;
Key:='\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders';
if Registry.KeyExists(Key) then
    begin
    Registry.OpenKey(Key,false);
    if Registry.ValueExists('Personal') then
       begin
       MyDocumentsDir:=Registry.ReadString('Personal');
       end;
    end;
```

```
Registry.CloseKey;
Registry.Free;
```

Uwaga! Aby usunąć tymczasowo powołany obiekt VCL zamiast wywoływać destruktor należy raczej wołać metodę Free (zob. help)

Ze względu na przejrzystość kodu pomijamy wszelką obsługę błędów.
Do kodu testującego naszą klasę można teraz dodać (po dodaniu do formy jeszcze jednego Labela) przypisanie `Label3.Caption:=Sfolder.MyDocumentsDir;`. Po uruchomieniu programu powinniśmy zobaczyć nazwy trzech interesujących nas katalogów.

Własność CurrentDir (property)
Własności obiektu tworzone z wykorzystaniem słowa kluczowego property pozwalają ukryć przed użytkownikiem szczegóły działania obiektu. Pozornie proste przypisanie lub odczytanie wartości własności oznacza w rzeczywistości uruchamianie odpowiednich skojarzonych z własnością metod.

Do deklaracji klasy TSFolders (w pliku nagłówkowym) dopiszmy następującą linię w sekcji public:

```
property WorkingDir :string read GetWorkingDir write SetWorkingDir;
```

Oznacza ona, że przy próbie odczytania wartości własności WorkingDir będzie uruchomiona metoda ReadWorkingDir, której wartość powinna być typu zadeklarowanej własności (tj. AnsiString), a próba zmiany wartości tej własności uruchomi metodę ChangeWorkingDir, której argumentem jest AnsiString. Musimy zatem zadeklarować i zaimplementować odpowiednie metody. Do sekcji private dopiszmy:

```
private
      function GetWorkingDir :string;
      procedure SetWorkingDir(AWorkingDir :string);
```

a w pliku SFolder.cpp zdefiniujmy następująco ciała tych metod:

```
function TSFolder.GetWorkingDir;
var tmpdir :array[0..MAX_PATH] of char;
begin
GetCurrentDirectory(MAX_PATH,tmpdir);
GetWorkingDir:=string(tmpdir);
end;

procedure TSFolder.SetWorkingDir(AWorkingDir :string);
begin
SetCurrentDirectory(PChar(AWorkingDir));
end;
```

W tej chwili, jeżeli użyjemy przypisania do lub czytania z własności `SFolder1.WorkingDir`, wykonane zostaną odpowiednie funkcje WinAPI Get/SetCurrentDirectory do odczytania lub zmiany bieżącego katalogu. Aby przetestować działanie tego fragmentu kodu możesz dodać TEdit lub TOpenDialog do zmiany katalogu i kolejny Label do jego pokazania.

Zadanie:
Przeciążyć konstruktor tak, aby mógł przyjmować jako argument nazwę nowego bieżącego katalogu.

2. Klasa TStoper

Bardzo łatwo stworzyć prosty stoper korzystając z komponentu TButton jako klasy bazowej.

Deklarujemy klasę TStoper:

```
type TStoper = class(TButton) //TButton wymaga stdctrls
     public
```

```
        constructor Create(AOwner: TComponent); override;
                                        //TComponent wymaga classes
    private
            miliseconds :Longint;
            Timer :TTimer; //TTimer wymaga extctrls
            procedure Start(Sender :TObject);
            procedure Stop(Sender :TObject);
            procedure Reset(Sender :TObject);
            procedure Tick(Sender :TObject);
    end;
```

zawierającą konstruktor typowy dla komponentów (argument, którym przekazuje się właściciela obiektu – nie mylić z rodzicem). W sekcji prywatnej umieszczona jest deklaracja zmiennej miliseconds, która będzie przechowywała ilość tysięcznych sekundy od naciśnięcia przycisku (TStoper dziedziczy z TButton, który jest po prostu przyciskiem), wkaźnik do TTimera („impulsowego zegara") oraz czterech metod, których argumenty są tak dobrane, żeby odpowiadały zdarzeniom OnClick w TButton i OnTimer w TTimer.

W konstruktorze przypisujemy własności geometryczne (domyślne, użytkownik może je zmienić) komponentu, napis na przycisku.

```
constructor TStoper.Create(AOwner: TComponent);
begin
Inherited Create(AOwner); //wywolanie konstruktora klasy bazowej TButton
Width:=200;
Height:=30;
Caption:='Start';
miliseconds:=0;

Timer:=TTimer.Create(Self);
Timer.Interval:=1;
Timer.Enabled:=false;
Timer.OnTimer:=Tick;

OnClick:=Start;
end;
```

Najważniejsze jest jednak stworzenie obiektu typu TTimer (jego adres przechowuje zadeklarowany w nagłówku wskaźnik Timer), ustalenie czasu między impulsami na jedną milisekundę oraz przypisanie zdarzeniu OnTimer metody Tick:

```
Timer.OnTimer:=Tick;
```

Widać, że przypisanie metody do zdarzenia polega na ustaleniu wartości odpowiednich wskaźników, którymi po prostu są zdarzenia, równej adresowi metod, z którymi mają być związane.
Metoda ta ma zwiększać naliczoną ilość milisekund oraz wyświetlać je na przycisku:

```
procedure TStoper.Tick(Sender :TObject);
var tmpstr :string;
begin
Inc(miliseconds);
Str(miliseconds,tmpstr);
Caption:=tmpstr+' ms';
end;
```

Ostatnim poleceniem konstruktora jest przypisanie zdarzeniu OnClick przycisku-stopera metody Start:

```
procedure TStoper.Start(Sender :TObject);
begin
Timer.Enabled:=True;
OnClick:=Stop;
end;
```

która uruchamia Timer i zmienia metodę zdarzeniową skojarzoną z OnClick na Stop.

Z kolei Stop:

```
procedure TStoper.Stop(Sender :TObject);
begin
Timer.Enabled:=False;
Caption:='Reset ('+Caption+')';
OnClick:=Reset;
end;
```

Wyświetla ostateczną ilość naliczonych milisekund i umożliwia wyzerowanie zegara zmieniając jeszcze raz metodę zdarzeniową na Reset, która pozwala na ponowne uruchomienie Stopera metodą Start:

```
procedure TStoper.Reset(Sender :TObject);
begin
Caption:='Start';
miliseconds:=0;
OnClick:=Start;
end;
```

Aby przetestować stoper należy powołać obiekt np. w konstruktorze Formy poleceniami:

```
procedure TForm1.FormCreate(Sender: TObject);
var Stoper1 :TStoper;
begin
Stoper1:=TStoper.Create(Self);
Stoper1.Left:=10;
Stoper1.Top:=10;
Stoper1.Parent:=Self;
end;
```

Zadanie
Zmienić format wyświetlania czasu na hh:mm:ss:ms dodając własności hours, minutes, seconds do grupy private.

3. Komponenty TLink i TMail

Należy pobrać z sieci plik link.pas lub stworzyć go za pomocą notatnika kopiując zawartość:

Plik link.pas

```
unit link;

interface

uses stdctrls,controls,classes,forms,graphics,shellapi,windows,dialogs;

type
TLink_base = class(TLabel)
   protected
      FAddress :string;
      procedure ColorBlack(Sender :TObject;Button :TmouseButton;
                           Shift :Classes.TShiftState;X :Integer;Y :Integer);
      procedure ColorNavy(Sender :TObject;Button :TmouseButton;
                           Shift :Classes.TShiftState;X :Integer;Y :Integer);
      procedure CheckAddress(_URL :string); virtual; abstract;
   public
      constructor Create(Owner :TComponent); override;
```

```
      procedure Connect(Sender :TObject);
   published
      property Address :string read FAddress write CheckAddress nodefault;
end;


TLink = class(TLink_base)
   private
      procedure CheckAddress(_URL :string); override;
   public
      constructor Create(Owner :TComponent); override;
end;


TMail = class(TLink_base)
   private
      procedure CheckAddress(_Address :string); override;
   public
      constructor Create(Owner :TComponent); override;
end;

procedure Register; //ta procedura obsluguje rejestrowanie komponentow

implementation

constructor TLink_base.Create(Owner :TComponent);
var LinkStyle :TFontStyles; //zbior
begin
   Inherited Create(Owner);
   LinkStyle:=[fsUnderline];
   Font.Color:=clNavy;
   Font.Style:=LinkStyle;
   OnClick:=Connect;
   OnMouseDown:=ColorBlack;
   OnMouseUp:=ColorNavy;
end;

procedure TLink_base.Connect(Sender :TObject);
begin
   ShellExecute(0,'open',PCHar(Address),'','',SW_NORMAL);
end;

procedure TLink_base.ColorBlack(Sender :TObject;Button :TmouseButton;Shift
:Classes.TShiftState;X :Integer;Y :Integer);
begin
   Font.Color:=clBlack;
end;

procedure TLink_base.ColorNavy(Sender :TObject;Button :TmouseButton;Shift
:Classes.TShiftState;X :Integer;Y :Integer);
begin
   Font.Color:=clNavy;
end;

//---------------------------------------------------------------------

constructor TLink.Create(Owner :TComponent);
begin
   Inherited Create(Owner);
   Caption:='http://www.phys.uni.torun.pl/~jacek';
   FAddress:=Caption;
```

```
   Hint:='Link to '+Caption;
   ShowHint:=true;
end;

procedure TLink.CheckAddress(_URL :string);
begin
   if Copy(_URL,1,7)<>'http://' then _URL:='http://'+_URL;
   FAddress:=_URL;
end;

//---------------------------------------------------------------------------

constructor TMail.Create(Owner :TComponent);
begin
   Inherited Create(Owner);
   Caption:='jacek@phys.uni.torun.pl';
   Address:='mailto:'+Caption;
   Hint:='Send a message to '+Caption;
   ShowHint:=true;
end;

procedure TMail.CheckAddress(_Address :string);
var hash,i :Integer;
begin
   if Copy(_Address,1,7)<>'mailto:' then
      begin
      _Address:='mailto:'+_Address;
      FAddress:=_Address;
      end;

   hash:=0;
   for i:=0 to Length(_Address) do if _Address[i]='@' then hash:=i;
   if hash=0 then ShowMessage('E-mail adress should include "@".')
             else FAddress:=_Address;
end;

//---------------------------------------------------------------------------

procedure Register;
begin
   RegisterComponents('JM', [TLink,TMail]);
end;

end.
```

Metody zdarzeniowe
W pliku nagłówkowym zadeklarowano trzy obiekty: TLink_base dziedziczący ze standardowej klasy VCL TLabel i dwie dziedziczące po nim klasy TLink i TMail.

Uwaga! Warto zauważyć, że konstruktory każdego komponentu wywołują konstruktory klas bazowych przed wykonaniem własnych specyficznych instrukcji.

Zadeklarowane w TLink_base metody ColorBlack i ColorNavy mają takie argumenty i wartość, aby mogły być metodami zdarzeniowymi zdarzeń OnMouseDown i OnMouseUp (tzn. mają identyczną postać jak funkcje tworzone przez środowisko dla tych zdarzeń dla TLabel). Podobnie metoda Connect jest dopasowana do zdarzenia OnClick. Przypisanie do odpowiednich zdarzeń znajduje się w konstruktorze TLink_base:

```
OnClick:=Connect;
OnMouseDown:=ColorBlack;
OnMouseUp:=ColorNavy;
```

Zadania tych metod są bardzo proste. Connect wywołuje funkcję ShellExecute (akcja systemu związana z rozszerzeniem pliku lub protokołem argumentu, w obiektach potomnych będą to protokoły http: i mailto:). Identyczną reakcję systemu można uzyskać korzystając z polecenia start w linii komend. Funkcje ColorBlack i ColorNavy zmieniają kolor napisu.

Metoda wirtualna / Klasa abstrakcyjna
W klasie bazowej (TLink_base) zadeklarowana jest metoda wirtualna CheckAdress. Dyrektywa virtual powoduje, że w klasie potomnej wywoływana jest odpowiednia metoda klasy potomnej, nawet jeżeli obiekt został powołany ze wskaźnikiem klasy bazowej. Dodatkowa dyrektywa abstract powoduje, że metoda staje się, posługując się językiem C++, pure virtual, i w ten sposób klasa staje się klasą abstrakcyjną - nie można deklarować obiektów tej klasy.

W klasach potomnych metody CheckAdress sprawdzają poprawność zmiennej prywatnej Faddress, w której przechowywany jest adres URL (w TLink) lub adresu e-mail (w TEmail). W pierwszym przypadku, w razie potrzeby, dodawany jest przedrostek „http://", w drugim dodawana jest nazwa protokołu „mailto:" i sprawdzana jest obecność „małpy" @.

Format napisu
W konstruktorze klasy TLink_base ustala się wygląd komponentu (podkreślenie, kolor, itp.) następującymi poleceniami (niezbędne jest zadeklarowanie zmiennej zbioru: `var LinkStyle :TFontStyles;`)

```
LinkStyle:=[fsUnderline];
Font.Color:=clNavy;
Font.Style:=LinkStyle;
```

W pierwszej linii zostaje zdefiniowany zbiór określający wygląd czcionki (możliwe elementy tego zbioru wyliczone są w TFontStyle). Jedynym elementem tego zbioru jest fsUnderline, tzn., że od normalnego wyglądu napis będzie różnił się jedynie podkreśleniem.

Własność Address
Najważniejszą nowością w tych komponentach jest opublikowana własność Address, którą można edytować za pomocą inspektora objektów. Wpisując poprawny adres e-mail w TMail (np. login@serwer.domena.pl) i adres WWW w TLink (np. www.domena.pl/~login) możemy sprawdzić działanie komponentów.

Rejestracja komponentu
Aby zarejestrować komponent należy do modułu Formy dodać procedurę (zadeklarowaną w sekcji interface):

```
procedure Register;
begin
RegisterComponents('JM', [TLink,TMail]);
end;
```

Uwaga! W razie niepowodzenia w rejestracji komponentu (np. zniknięcie wszystkich komponentów VCL i/lub błąd przy uruchomieniu Delphi) należy w katalogu LIB podmienić wadliwy plik dclusr30.dpk przez jego backup dclusr30.~dp.

Uwaga! Komponenty napisane w Object Pascalu mogą być kompilowane i rejestrowane w C++ Builder.

Ikony komponentów muszą znaleźć się w pliku **link.dcr** (ich nazwy muszą być identyczne z nazwami klas, muszą to być bitmapy o rozmiarach 24 x 24 pixeli)

Zadanie:
Zdarzenia OnClick, OnMouseDown i OnMouseUp mogą być zdefiniowane przez użytkownika po położeniu TLink lub TMail na formę, co powoduje zastąpienie ustawień z konstruktora. Należy usunąć te zdarzenia z grupy __published i ukryć je w ten sposób w Object Inspectorze.

4. Komponenty TLink i TMail – drobne modyfikacje

Stworzony przez nas komponent TLink można na kilka sposobów poprawić:
1) Należy usunąć własności (np. Caption, Color i kilka innych) i zdarzenia (wszystkie) do których programista nie powinien mieć dostępu.
2) Dodać własności (poza obecnym już Address) Description, ActiveColor, InactiveColor.

Ad. 1) W Delphi przy dziedziczeniu z komponentu nie można ukrywać własności ani zdarzeń (przenosić je do sekcji `protected` lub `private`). Możliwa jest tylko czynność odwrotna (można podawać deklaracje do sekcji `published` lub `public`). Programiści oryginalnego VCL ułatwili na szczęście życie twórcom komponentów tworząc komponenty pośrednie np. TCustomLabel, który posiada niemal pełną funkcjonalność TLabel, ale jego metody i własności zadeklarowane są w sekcji `protected`.
W tej sytuacji, aby ukryć zbędne metody i zdarzenia musimy zmienić klasę bazową na `TCustomLabel`:

```
TLink_base = class(TCustomLabel)
```

W tej chwili po zarejestrowaniu obiekt nie posiadałby większości własności (poza odziedziczonymi z TControl) i wszystkich zdarzeń. Część z nich znowu możemy udostępnić dodając do sekcji __published:

```
published
      property Align;
      property Enabled;
      property Font;
      property Transparent;
      property Visible;
```

Pozostałe pozostawimy w ukryciu ustalając ich wartość w konstruktorze:

```
ShowAccelChar:=false;
WordWrap:=false;
```

Ad 2) Dodamy również kolejne własne własności:

```
protected
      FDescription :String;
      FInactiveColor :TColor;
      FActiveColor :TColor;

published
      property Description :String read FDescription
                                    write SetDescription
                                    nodefault;
      property InactiveColor :TColor read FInactiveColor
                                      write SetColor
                                      nodefault;
      property ActiveColor :TColor read FActiveColor
                                    write FActiveColor
                                    nodefault;
```

`InactiveColor` i `ActiveColor` powinny być aktywowane w konstruktorze:

```
FInactiveColor:=clNavy;
FActiveColor:=clBlack;
```

Metody `SetDescription()` i `SetColor` powinny wyglądać następująco:

```
procedure TLink_base.SetColor(AColor :TColor);
begin
FInactiveColor:=AColor;
Font.Color:=FInactiveColor;
```

```
end;

procedure TLink_base.SetDescription(ADescription :String);
begin
FDescription:=ADescription;
Caption:=FDescription;
end;
```

Należy również zastąpić metody `ColorNavy()` i `ColorBlack()` przez

```
procedure TLink_base.ChangeColorActive(Sender :TObject;
    Button :TmouseButton;Shift :Classes.TShiftState;X :Integer;Y :Integer);
begin
Font.Color:=FActiveColor;
end;

procedure TLink_base.ChangeColorInactive(Sender :TObject;
    Button :TmouseButton;Shift :Classes.TShiftState;X :Integer;Y :Integer);
begin
Font.Color:=FInactiveColor;
end;
```

Uwaga!
Istotną rzeczą jest taka modyfikacja konstruktorów klas potomnych `TLink` i `TMail`, aby nie odpwoływały się do własności `Caption`, a zamiast tego, żeby ustalały własność `Description`.

5. Zdarzenia komponentu TStoper (zadanie)

Należy przygotować komponent `TStoper`. Poza funkcją rejestrującą należy dodać do niego zdarzenia `OnStart`, `OnStop` i `OnReset`. Zdarzenia deklaruje się analogicznie jak własności, z tym, że nie odwołują się do metod komponentu, a do prywatnego wskaźnika do metody zdarzeniowej (`TNotifyEvent` jest takim wskaźnikiem), z którą może ją powiązać programista. Innymi słowi zdarzenie jest formą przekazywania wskaźnika metody do obiektu.

```
//Zdarzenia
private
      FOnStart :TNotifyEvent;
      FOnStop :TNotifyEvent;
      FOnReset :TNotifyEvent;
published
      property OnStart :TNotifyEvent read FOnStart write FOnStart;
      property OnStop :TNotifyEvent read FOnStop write FOnStop;
      property OnReset :TNotifyEvent read FOnReset write FOnReset;
```

Przed wywołaniem metody zdarzeniowej należy sprawdzić czy została podłączona. Inaczej takie wywołanie skończy się to błędem.

```
procedure TStoper.Start(Sender :TObject);
begin
Timer.Enabled:=True;
OnClick:=Stop;
if Assigned(OnStart) then FOnStart(Self);
end;
```

VIII. Korzystanie z gotowych komponentów

1. Skąd wziąć gotowe komponenty?

Np.
DSP: http://sunsite.icm.edu.pl/delphi
Torry: http://www.torry.ru

2. Instalowanie gotowych komponentów

Komponenty (i moduły) pisane w Delphi można kompilować i rejestrować w C++ Builderze. Odwrotnie niestety nie jest (jedyny sposób na wykorzystanie kodu napisanego w C++ Builderze pod Delphi bez jego tłumaczenia to przygotowanie bibliotek DLL).

W C++ Builderze 1 w **menu Component** znajdziemy pozycję **Install....** Tu powinniśmy kliknąć klawisz **Add...**, następnie **Browse** i musimy wkazać plik *.cpp, *.pas lub *.obj (+ *.h) zawierający komponent, który chcemy zainstalować.
Uwaga! Podobnie jak w przypadku własnoręcznie napisanego komponentu instalacja może się nie powieść i pomocne może być przywrócenie poprzedniej wersji biblioteki (zob. uwagi przy tworzeniu komponentów).

Komponenty w nowszych wersjach C++ Buildera i Delphi są grupowane w pakietach (packages). Można w analogiczny sposób jak w wersji 1.0 zainstalować pojedyńczy komponent do nowego lub wybranego pakietu. Można również instalować cały pakiet (jak widzieliśmy pisząc komponenty funkcja rejestrująca przyjmuje w argumencie listę komponentów i może być w jednym pliku wywołana wieloktotnie).

Zadanie
Odnajdź na sieci i zainstaluj następujące komponenty:
1. TrayIcon (Shellage),
2. GProgress,
3. MarsCaption.

IX. Gdzie szukać pomocy w sprawach C++ Buildera i Delphi?

W kolejności:

1. Uruchomić help (termin z którym mamy problem + F1)
2. Jeżeli problem dotyczy np. WinAPI należy zajrzeć do MS Help (w starszych wersjach spis zawartości nie był zintegrowany z ogólnym helpem)
3. Zajrzeć do książek (np. *Delphi 2. Vademecum profesjonalisty*)
4. Sprawdzić czy potrzebne nam rozwiązania znajdują się w przykładach dostarczonych razem z Delphi (katalog DEMOS) i C++ Builderem (EXAMPLES)
5. Sprawdzić czy problem jest opisany w którymś FAQ (np. na stronie grupy dyskusyjnej pl.comp.lang.delphi, czyli http://delphi.koti.com.pl)
6. Przeszukać sieć światową
7. Wysłać pytanie na pl.comp.lang.delphi lub pl.comp.lang.c (jeżeli pytanie ma oczywistą odpowiedź należy się spodziewać niemiłej reakcji)

Dodatek A:
Tłumaczenie kodu pomiędzy Delphi i C++ Builderem

Wszystkie fragmenty kodu źródłowego, które znacząco różnią się w Object Pascalu i C++ (np. obsługa zbiorów) podane są w obu wersjach. Natomiast proste przypisywanie własności lub wywoływanie funkcji i metod często podane jest tylko w Pascalu lub tylko w C++. Tłumaczenie jest jednak bardzo proste. Najczęściej powtarzające się sekwencje zostały zebrane w poniższej tabeli.

C++ Builder	Delphi	wyjaśnienie

`{`
`...`
`}`
	`begin`
	`...`
	`end;`
		początek i koniec sekcji

`a=1`	`a:=1`	operator przypisania
`if (i==1) ...; else ...;`	`if i=1 then ... else ...`	instrukcja warunkowa
`select (zmienna)` `{case ... }`	`case zmienna of`	instrukcja wyboru (więcej w helpie)
`for (i=0; i<=10; i++)` `...`	`for i:=0 to 10 do ...`	najprostsza postać pętli
`obiekt->metoda` *`obiekt->własność`*	*`obiekt.metoda`* *`obiekt.własność`*	Obiekty VCL w C++ Builderze są wskaźnikami, natomiast w Delphi nie. Dlatego w C++ wymagany jest operator *obiekt->własność* (*member selector*) równoważny *(*obiekt).własność*.
`zmienne mogą być deklarowane gdziekolwiek`	*`zmienne deklarowane są jedynie w sekcji var`*	
`Klasa zm = new Klasa();`	*`var zm :Klasa;`* *`begin`* *`zm:=Klasa.Create()`*	Tworzenie obiektów VCL Konstruktor w C++ ma nazwę identyczną jak nazwa klasy, a w Delphi zazwyczaj jest to Create
`/* ... */`	*`{ ... }`*	Komentarze

Nazwy odpowiadających sobie zmiennych można znaleźć w pomocy do C++ Buildera w temacie „Table of Delphi data types”:

Delphi	Size/Values	**C++ implementation**	Implementation
ShortInt	8-bit integer	char	typedef
SmallInt	16-bit integer	short	typedef
LongInt	32-bit integer	**long**	typedef
Byte	8-bit unsigned integer	unsigned char	typedef
Word	16-bit unsigned integer	unsigned short	typedef
Integer	32-bit integer	**int**	typedef
Cardinal	32-bit unsigned integer	unsigned long	typedef
Boolean	true/false	**bool**	typedef
ByteBool	true/false or 8-bit unsigned integer	unsigned char	typedef
WordBool	true/false or 16-bit unsigned integer	unsigned short	typedef
LongBool	true/false or 32-bit unsigned integer	unsigned long	typedef
AnsiChar	8-bit unsigned character	unsigned char	typedef
WideChar	word-sized Unicode character	wchar_t	typedef
Char	8-bit unsigned character	**char**	typedef
AnsiString	Delphi AnsiString	AnsiString	class
String[n]	old style Delphi string, n = 1..255 bytes	SmallString<n>	template class
ShortString	old style Delphi string, 255 bytes	SmallString<255>	typedef
String	Delphi AnsiString	**AnsiString**	typedef

Single	32-bit floating point number	float	typedef
Double	64-bit floating point number	**double**	typedef
Extended	80-bit floating point number	long double	typedef
Currency	64-bit floating point number, 4 decimals	Currency	class
Real	32-bit floating point number	**float**	typedef
Comp	64-bit floating point number	double	typedef
Set	1..32 bytes	Set<type, minval, maxval>	template class
Pointer	32-bit generic pointer	**void ***	typedef
PChar	32-bit pointer to characters	**unsigned char ***	typedef
PAnsiChar	32-bit pointer to ANSI characters	unsigned char *	typedef
Variant	OLE variant value (16 bytes)	Variant	class